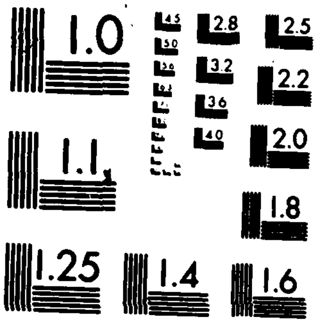
AD-A130 885 DYNAMIC PROGRAMMING ALGORITHMS AND ANALYSES FOR NONSERIAL NETWORKS PART I..(U) GEORGIA INST OF TECH ATLANTA SCHOOL OF INDUSTRIAL AND SYSTEMS.. A O ESOGBUE 1/2

UNCLASSIFIED JAN 83 ARO-17672.1-MA-H DAAG29-80-G-0010 F/G 12/2 NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

12

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
17672.1-MA-H

2. GOVT ACCESSION NO.
AD-A130885

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Dynamic Programming Algorithms and Analyses for Nonserial Networks: Part I

5. TYPE OF REPORT & PERIOD COVERED
Final:
25 Sep 80 - 24 Sep83

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Augustine O. Esogbue

8. CONTRACT OR GRANT NUMBER(s)
DAAG29 80 G 0010

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Georgia Institute of Technology
Atlanta, GA 30332

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
U. S. Army Research Office
Post Office Box 12211
Research Triangle Park, NC 27709

12. REPORT DATE
Jan 83

13. NUMBER OF PAGES
106

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

computer programming
nonserial networks
operations research
dynamic programming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report discusses the research conducted by the research team at the Georgia Institute of Technology in the area of nonserial dynamic programming networks. The problems, approach and major results are summarized in Chapter 1, while the rest of the chapters discuss, in detail, the algorithm developments, experiments with sample problems, and algorithmic complexities. Each chapter contains detailed computer flow charts for the algorithms developed. Chapter 6 employs an efficient dimensionality reduction algorithm known as the imbedded state space method in conjunction with the one developed in this study to treat an otherwise intractable

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (cont.)

problem involving feedforward loop systems. These algorithms are useful inputs to the development of dynamic programming based strategies for the analysis of complex nonserial networks.

successful attempts to generalize or develop a theory applicable to most, if not

DYNAMIC PROGRAMMING ALGORITHMS AND ANALYSES

FOR

NONSERIAL NETWORKS: PART I

By

Augustine O. Esogbue

Completion Report

GTRI Project Nos. E-24-623 and 645

ARO No. DAAG 29-80-G-0010

Initiated: October 1980

Completed: January 1983

The work upon which this report is based was supported by the U.S. Army Research Office and the Atlanta University through a subcontract to the Georgia Institute of Technology, Atlanta, Georgia.

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

ACKNOWLEDGEMENTS

This study was conducted under a subcontract from the Atlanta University as part of the U.S. Army Research Office Prime Grant No. DAAG 29-80-G-0010 (GTRI Project No. E-24-623 and E-24-645). We are grateful to Drs. Chandra and Boggs of the Mathematics Division, USARO for considering and funding the project. We are also grateful to the officials of the Atlanta University and Georgia Tech for making it possible for Dr. Warsi and me to participate in what was originally a joint research effort. Dr. Warsi contributed to the development of the methods of complexity analysis which were used in the algorithmic analyses throughout this report.

I am very grateful to my doctoral student, Mr. Chaeyoung Lee for programming and computational assistance. The report benefitted from the diligent typing of Ms. Pamela Singleton, my secretary, who also coordinated other aspects of the project.

Augustine O. Esogbue, Ph.D.
Professor and Principal Investigator
Georgia Institute of Technology

ABSTRACT

This report discusses the research conducted by the research team at the Georgia Institute of Technology in the area of nonserial dynamic programming networks. The problems, approach and major results are summarized in Chapter 1, while the rest of the chapters discuss, in detail, the algorithm developments, experiments with sample problems, and algorithmic complexities. Each chapter contains detailed computer flow charts for the algorithms developed. Chapter 6 employs an efficient dimensionality reduction algorithm known as the imbedded state space method in conjunction with the one developed in this study to treat an otherwise intractable problem involving feedforward loop systems. These algorithms are useful inputs to the development of dynamic programming based strategies for the analysis of complex nonserial networks.

TABLE OF CONTENTS

CONTENTS (CONTINUED)

FIGURES

TABLES

Chapter 1

NONSERIAL DYNAMIC PROGRAMMING NETWORKS: THE PROBLEM, APPROACH, AND MAJOR RESULTS

1.1 Introduction

Our overall interest is in the optimal analysis (and/or design) of large scale systems. Generally, a large complex system is composed of several interconnected subsystems which individually may be simpler than the parent system. In nonserial systems, the structure of these interconnections creates further complexity. Such complexities, for example, may be engendered by the presence of combinations of various nonserial networks.

A nonserial system, as defined by Beightler and Meier [1], is a system where at least one subsystem in the system receives inputs from more than one sub-subsystem or sends outputs to more than one subsystem. It could also be described as a system where for at least one of the stages, the output is not the input to the next; thus, there exists at least one n such that the output $x_n \neq x_{n-1}$, the input of the next stage. Nonserial systems are encountered in the study of chemical processing systems, natural gas transmission pipelines, water resources systems, energy, production-inventory systems, and various other systems. Practical examples of these systems are further discussed by Esogbue and Marks [14]. Thus, there exist important reasons to study such systems.

To motivate our discussion, let us consider a general mathematical formulation for the following nonserial system, which approximates our concept of a large scale system.

1.2 A General Nonserial System: Some Motivations

For illustrative purposes, we introduce the following general complex nonserial system which is an example of a complex network:

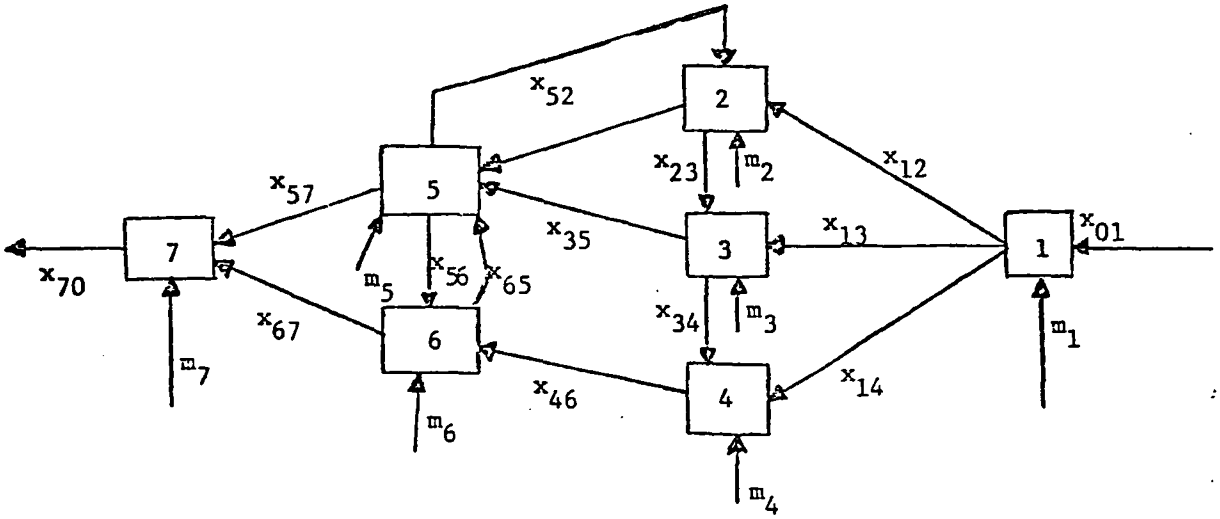


Figure 1.1: An Example of a Large Complex Network

In the foregoing, let

Y_i be the vector of inputs to subsystem i, and

M_i, the corresponding vector of decision variables, $m_i \varepsilon M_i$.

Further, let Z_i be defined as the set of output variables to subsystem i, with z_i defined as $z_i \varepsilon Z_i$, and $f_i(Y_i, M_i)$ as the objective function for subsystem i. In this representation, we may visualize

$z_i = \Phi(Y_i, M_i)$ as the set of interconnection relations for subsystem i and $(Y_i, M_i) \varepsilon S_i$, where S_i is the set of restrictions for subsystem i. Note the generality of these definitions i.e., $f_i(.,.)$ and Φ $(.,.)$ need not be familiar functions.

As an example, consider subsystem 3: $Y_3 = \{x_{23}, x_{13}\}$; $Z_3 = \{x_{35}, x_{34}\}$. A mathematical programming formulation for this nonserial system is the following:

$$\max \sum_{i=1}^{7} f_i(Y_i, M_i) \tag{1}$$

$$\text{s.t. } z_i = \Phi(Y_i, M_i) \ \forall \ z_i \ \varepsilon \ Z_i \qquad i = 1, 2, \ldots, 7 \tag{2}$$

$$(Y_i, M_i) \ \varepsilon \ S_i \qquad i = 1, 2, \ldots, 7 \ . \tag{3}$$

Note that this complex system contains subsets of various classical nonserial systems such as diverging branch, converging branch and feedback loop systems, and as such may be considered a generalized network. Its treatment is thus nontrival.

When the objective function and the constraints are linear, the resulting mathematical programming problem can easily be solved via linear programming methods. When the objective function and the constraints are convex functions defined over a convex set, the mathematical programming problem becomes a convex programming problem for which there are methods of solution. However, when any of the convexity assumptions are dropped, this mathematical programming problem becomes difficult to solve. Dynamic programming does not depend upon the nature of the objective function, constraints, or the construct of the domain of the feasible region of search. It thus possesses some appeal for solving problems of this genre.

Dynamic programming has been used to optimize nonserial systems in various areas. Wong and Larson [27] for instance, used dynamic programming in the design and operation of natural gas transmission pipelines with a diverging branch structure. Mitten and Nemhauser [19] applied this method to a hypothetical chemical process which contained a recycle feedback loop around a reactor. Beightler and Meier [2] considered the application of dynamic programming to a river basin reservoir system with a converging branch structure. Esogbue and Marks [15] studied several project scheduling and resource allocation problems of the CPM-Cost variety in which the precedence relationships possess a nonserial structure. Some efficient dynamic programming based procedures for network compression were advanced.

Obviously, more real life systems can be formulated as nonserial dynamic programming problems. The limited invocation of the method for the analysis of more complex structural systems is attributable, in the main, to the lack of a sufficiently "n^{th}" order theory of nonserial systems and to the usual computational problems that have plagued the application of dynamic programming. Methods for alleviating the computational burden so frequently encountered can be found in another paper by Esogbue and Marks [14].

1.3 Previous Related Works

A number of contributors to the state of knowledge of nonserial dynamic programming deserve mention. Wilde and Beightler [26] and Nemhauser [21] reviewed the basic theory involved in the optimization of the four basic classes of nonserial systems: 1) diverging branch systems, 2) converging branch systems, 3) feedforward loop systems, and 4) feedback loop systems. More complicated nonserial systems, however, cannot be studied unless the results for the previously mentioned four systems can be applied to the problem. The example in Section 2, for instance, cannot be optimized efficiently using the method of nonserial dynamic programming contained in the the original works cited above. Simple modifications and extensions of the theory are not helpful either.

One reason for the above quandry is simply that this system is not composed of a simple combination of the above mentioned four classes of nonserial systems. We are thus unable to determine the order in which the subsystems should be optimized, so that the dynamic programming procedure can be performed efficiently.

Another important reason is that computational techniques capable of being used to tackle the nonserial examples given in the literature do not exist. Furthermore, the issues are not discussed anywhere. One is left to impute that traditional computational problems inhibiting widespread use of dynamic programming become exacerbated in the nonserial case.

Among the major contributors to the current literature of dyanmic programming are Bertele and Brioschi [5, 6, 7, 8]. However, their main concern is with the optimization of a problem whose objective function has the following specialized form:

$$\min_{X} F(X) = \min_{X} \sum_{i \varepsilon T} f_i(X^i) \tag{4}$$

in which

$X = \{x_1, x_2, \ldots, x_n\}$ is a set of discrete variables, σ_i being the number of feasible values of the variable x_i. (5)

$T = \{1, 2, \ldots, t\}$ and $X^i \subset X$. (6)

In the above, the function F(X) is called the objective function and the functions $f_i(X^i)$ are the components of the objective function. Before Bertele and Brioschi can optimize this problem via dynamic programming, the order in which the components of the objective function are to be optimized by the dynamic programming procedure has to be determined, so that the number of computations for the problem can be minimized. A series of algorithms using graph theory concepts to determine this optimal order is then developed. While these contributions are important, the limitations of the problem addressed and thus the algorithms are evident. The other contributors to the computational aspects of nonserial dynamic programming include Beightler, Johnson and Wilde [1], Parker [24], Parker and Crisp [25] and Brown [9]. The superposition approach suggested by Beightler et al [1] for treating converging branch systems under the assumption of linear return and transition functions with additive compositor operator was extended to nonlinear converging branch systems by Parker and Crisp [25]. An extension of some of these concepts to feedforward and feedback loop systems was presented by Parker. In [9] Brown considers a different approach to the analysis of converging branch systems under both deterministic and stochastic return and transition functions. The procedure considers the nonserial converging branch system as a serial system by grouping the stages of the two branches together into new two dimensional stages. In this case,the input and output state vectors, as well as the decision vectors, are all two dimensional. The approach, although apparently demanding on the storage requirement, has interesting features especially with regards to the analysis of stochastic systems.

Because of our concern for complexity reduction, we have demonstrated in [15] that an adroit combination of certain potent but hitherto isolated concepts and techniques of large scale problem solving can lead to the solution, via nonserial dynamic programming, of certain interesting nonserial systems. Specifically, efficient formulations for treating a three branch converging system, a system with multipaths departing from a junction, and a complex converging-diverging-converging system, were developed by an adroit synthesis of the pseudo stage concept (Beightler and Meier [2]), Nemhauser and Ullman's method [23] and an optimal elimination technique akin to Bertele and Brioschi [6]. This method proved to be considerably more efficient than the solo application of any of the foregoing or any currently available algorithm.

1.4 Research Objectives

The overall objective of this research is to extend the theory of nonserial dynamic programming so that it can be applied to most nonserial systems. In [12], we demonstrated that the following three factors hinder the solution of a problem by dynamic programming: 1) the amount of high speed memory required for the problem (space complexity), 2) the total number of calculations required (computational complexity), and 3) the amount of off-line memory required (space complexity). With these criteria in mind we wish to develop an algorithm which, in addition, will give an optimal order in which the subsystems in any nonserial system should be optimized.

This is based on the premise that the crucial issue in nonserial systems research revolves around the set of questions: a) given a complex nonserial system, how do we collapse it into a serial-like structure? b) in the branch compression effort, what optimal order should be followed in order for the resultant dynamic program to be efficient with regards to the usual issues in dynamic programming algorithm development? While Bertele and Brioschi have proposed an optimal compression order for a highly specialized nonserial form, we are unaware of any successful attempts to generalize or develop a theory applicable to most, if not all, nonserial forms. While we would like to move in this direction, we thought that our initial efforts should be directed to the development of efficient algorithms for the treatment of the classical nonserial systems which link up, on a higher level, to form a complex nonserial system. These algorithms would be accompanied with detailed treatments of their algorithmic complexities. Such analyses were particularly missing in previous works in the literature. They are, however, considered absolutely necessary in the development of strategies for resolving complex nonserial network problems.

1.5 Research Plan and Results

The research consisted of the following six phases:

1.5.1) Phase 1

A number of criteria for discussing efficiency exist. These are, in turn, contingent on the choice of the objective function. In the first phase, we considered these issues and determined a set of criteria for which we wished to develop an algorithm. They were based upon the three principal factors which tend to hinder the solution of a problem via dynamic programming as well as those in combinatorics. Several criteria were examined first before focusing on the most appropriate set or combinations. Structural characteristics (attributes) of a complex nonserial system of the type depicted in Fig. 1.1 were studied leading to a grouping of complex nonserial systems by attributes and by degree of complexity. For example, we developed a characterization of the complexity of a nonserial network in terms of the following parameters: N, the number of nodes; M, the connectedness, and ordering or arc orientation such as diverging, converging, feedforward, and feedback and various combinations of each.

Using these parameters, the complexities may be described as follows:

1) Large N and simple structure

2) Large N and complex combination of arc orientations such as converging branch, diverging branch, feedforward and feedback loops.

3) Small N and simple structure

4) Small N and complex structure

1.5.2) Phase 2

An algorithm was then developed for each of the criteria decided upon in Phase 1. The algorithms were structured so that the order of optimization of the subsystems of the classical nonserial systems was the best possible. We invoked concepts from graph theory, signal flow graphs, and automata theory wherever possible in the dynamic programming algorithm development.

Using the conventional DP algorithm, a computer code in FORTRAN V was developed for both the diverging and converging branch systems. Next, detailed computer algorithms for these systems were constructed. Using algorithm analysis, results were derived to describe the space and time complexities of these algorithms. For example, if we let the discretization levels for the state and decision variables be denoted by K_S and K_D respectively, and further let M be the number of stages in the subbranch while N represents the number in the main serial system, then we obtain the following results:

a) For the Diverging Branch System

The maximum space requirement = $(M+N+3)K_S$, and

The number of computations = $(M^2+N^2+M+N) \quad (K_D+1)K_S + 4(M+N)+3K_S + 9$

b) For the Converging Branch System

The maximum space requirement = $2K_S^2 + (MK_D+ N+3)K_S$, and

The number of computations = $[(K_D+1) \; (M^2+N^2+M+N) + 4] \; K_S^2 + (K_D+4)K_S + 4M + 4N - 1$

These derived results are important in many ways. For example, they may be used to compare the computational and space complexities of classical nonserial networks under the following restrictions:

i) identical number of branches

ii) identical number of nodes in subbranch as well as in the main serial system, i.e. $N = 2M$

iii) discretization levels for states and decisions are the same, i.e. $K_D = K_S$.

The following comparisons are instructive:

a) <u>Space complexity comparison</u>:

Space for diverging branch structure = $3(M+1)K_S$

Space for converging branch structure = $(M+2)K_S^2 + (2M+3)K_S$

b) <u>Computational complexity comparison</u>:

Number of computations for diverging branch structure =

$M(5M+3)K_S^2 + (5M^2+3M+3)K_S + 8M + 9$

Number of computations for converging branch structure =

$M(5M+3)K_S^3 + (5M^2+3M=5)K_S^2 + 4K_S + 12M - 1$.

We sum up the comparisons as follows: (See Table 1.1)

Table 1.1: Comparison of Complexities of Diverging and Converging Branch Systems

Structure	Complexity Name	Complexity as a function of	Nature of Complexity
Diverging branch	Space	Number of nodes in the branch	Linear
Converging branch	Space	Number of nodes in the branch	Linear
Diverging branch	Space	Discretization levels	Linear
Converging branch	Space	Discretization levels	Quadratic
Diverging branch	Computational	Number of nodes in the branch	Quadratic
Converging branch	Computational	Number of nodes in the branch	Quadratic
Diverging branch	Computational	Discretization levels	Quadratic
Converging branch	Computational	Discretization levels	Cubic

We may thus conclude that both from space and computational complexity considerations, the diverging branch system is less complex than the converging branch one - a fact that is suggested by the problem structure. Detailed analysis and results appear in the respective chapters.

1.5.3) Phase 3

High level computer programs in FORTRAN V were written for both the diverging branch and converging branch systems. These were exemplified via several test problems. The systems were subjected to perturbations in both N and M and return as well as transition functions. Their algorithmic complexities were experimentally verified in each case. Refinements in the algorithms were executed by introducing concepts such as branch compression and node elimination, especially for complex multi-branch systems. Detailed computer flow charts for each problem were constructed. We emphasize the utility of our approach. The algorithms admit input, return, and transition data in terms of functions of the associated variables. Storage problems were not encountered with this line of pursuit. For programs involving data structures in the form of tables, see the report by Dr. N. Warsi.

1.5.4) Phase 4

In this phase, the exercises of phases 2 and 3 were extended to the more complex systems namely the feedforward and feedback loop systems.

1.5.5) Phase 5

Because of our interest in efficiency of algorithms, we explored the possibility of an adroit combination of algorithms to solve problems which might otherwise defy solution via a solo application of an algorithm. Thus, we constructed a feedforward loop example in which the return functions and the constraints possess a special structure. In particular, the problem had components which had the characteristics of a multidimensional nonlinear knapsack problem. This problem, as posed, could not be solved without major modifications in our algorithm. It was, however, successfully solved using a combination of our algorithm and the imbedded state space dynamic programming routine.

1.5.6 Phase 6

Finally, we considered the following assertion which we developed in connection with our study of nonserial networks of the CPM-Cost variety using the project cost minimization criterion:

> *Whenever a network is such that for each activity i and j, with activity i preceding activity j $(i < j)$ the set of paths p_i and p_j containing activities i and j respectively is given by $p_i \subseteq p_j$ or $p_j \subseteq p_i$ then we can assert that the functional equation $f_n(L)$ decomposes into a sequence of one dimensional dynamic programming problems.*

This is a sufficient condition which we can prove rather easily. We attempted to develop a condition that is both sufficient and necessary with the hope that an algorithm useful in the analysis of more complex systems may result. However, because of time and other resource constraints, this line of pursuit was truncated for the moment.

1.6 Report Outline

Having discussed the project background, purpose and major results in this chapter, we dedicate the rest of the report to the development of the dynamic programming analysis of the various classical nonserial systems. We begin with the diverging branch system, the easiest of the systems to analyze. This is treated in Chapter 2. The algorithm, flowcharts, sample problems and algorithmic analyses are given. In Chapter 3, the converging branch system is treated. This chapter includes two algorithms - the original and a modified one for the purpose of solving problems with complex functions. In Chapter 4, nonserial systems with feedforward loop structures are discussed. The loop system is very complex since it involves a combination of the diverging and converging branch systems. Chapter 5 considers the feedback loop nonserial problem. This is akin to the feedforward in many ways. The report is concluded with Chapter 6. In this chapter, a feedforward loop example was constructed involving return and transition functions that were different from any type considered previously. In particular, the return functions were step functions and the resource availability constraints were expressed in terms of constants. The imbedded state space algorithm was used in conjunction with our algorithm to solve this problem. The algorithmic steps are provided but computer implementation was truncated because of budgetary and time constraints on this project. It could be seen though that this is a beneficial and necessary line of pursuit when structurally complex nonserial systems are being investigated.

Chapter 2

ANALYSIS OF DIVERGING BRANCH NONSERIAL NETWORKS

2.1 Development of a Dynamic Programming Algorithm for the Diverging Branch System

A diverging branch system (see Fig. 2.1) is the easiest of the elementary nonserial structures to analyze. For simplicity, we first consider a two branch system. The stage transformations and return functions both for a main serial process and for a branch are defined as follows:

$$x_{n-1} = t_n(x_n, d_n) \quad , \quad n = 1, 2, \ldots, N$$

$$x_{m-1,1} = t_{m1}(x_{m1}, d_{m1}) \quad , \quad m = 1, 2, \ldots, M$$

and

$$x_{M1} = t_{S1}(x_S, d_S)$$

$$r_n = r_n(x_n, d_n) \quad , \quad n = 1, 2, \ldots, N$$

$$r_{m,1} = r_{m,1}(x_{m1}, d_{m1}) \quad , \quad m = 1, 2, \ldots, M$$

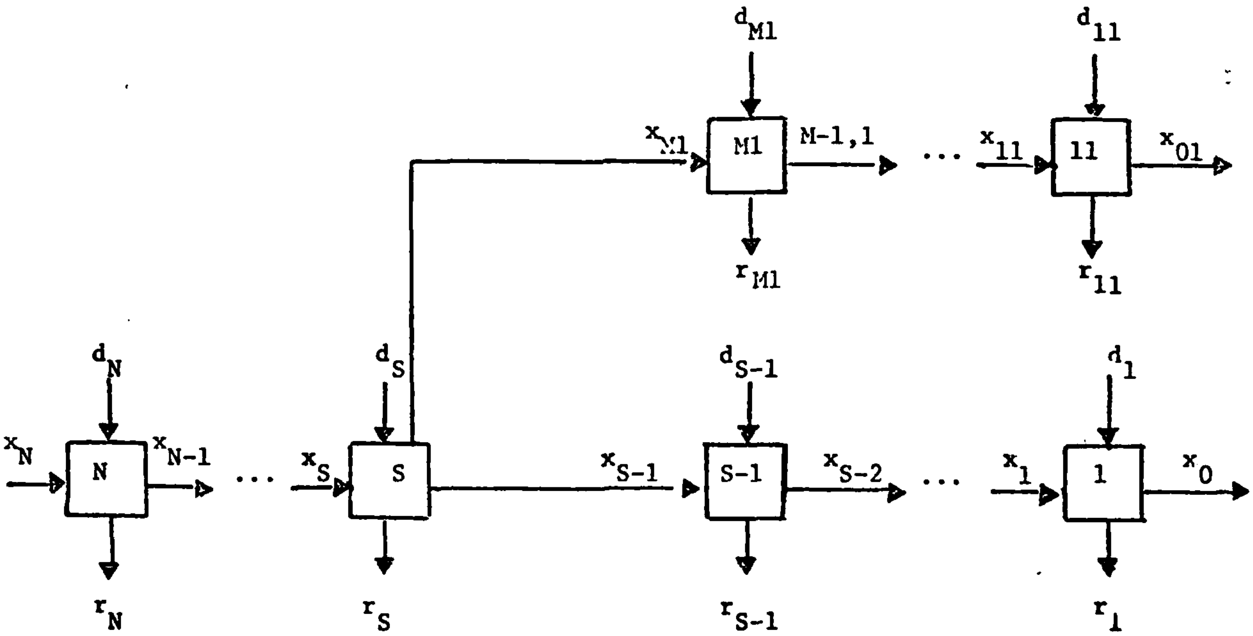


Figure 2.1: A Diverging Branch System

Consider the basic system consisting of one main serial system ($n = 1, 2, \ldots, N$) and one branch ($M = 1, 2, \ldots, M$). Let us assume that the input and decision variables at each stage have the following integer values:

$$
\begin{array}{ll}
1 \leq x_{i1} \leq K_{i1} & ,\quad i = 1, 2, \ldots, M \\
1 \leq x_{i} \leq K_{i} & ,\quad i = 1, 2, \ldots, N \\
1 \leq d_{i1} \leq P_{i1} & ,\quad i = 1, 2, \ldots, M \\
1 \leq d_{i} \leq P_{i} & ,\quad i = 1, 2, \ldots, N
\end{array}
$$

To develop our algorithm, we first decompose the network into four phases and then we employ the usual recursive procedures in optimizing the total return. The recursion equations for the various phases are defined as follows:

2.1.1) For the Diverging Branch (from stage 11 to stage M1)

$$f_{11}(x_{11}) = \max_{1 \leq d_{11} \leq P_{11}} r_{11}(x_{11}, d_{11})$$

$$f_{m1}(x_{m1}) = \max_{1 \leq d_{m1} \leq P_{m1}} [r_{m1}(x_{m1}, d_{m1}) + f_{m-1,1}(t_{m1}(x_{m1}, d_{m}))]$$

where $m = 2, 3, \ldots, M$

Using the above equations, the optimal branch return and optimal decisions are computed for each possible value of x_{M1}.

2.1.2) For the Main Serial Process (from stage 1 to stage S-1, (prior to junction node)

$$f_1(x_1) = \max_{1 \leq d_1 \leq P1} r_1(x_1, d_1)$$

$$f_n(x_n) = \max_{1 \leq d_n \leq P_n} [r_n(x_n, d_n) + f_{n-1}(t_n(x_n, d_n))]$$

where $n = 1, 2, \ldots, S-1$

The optimal return $f_n(x_n)$and optimal decision d_n at each stage are saved for each possible input value x_n:

2.1.3) For the Stage S (junction)

$$f_{S+M1}(x_S) = \max_{1 \leq d_S \leq P_S} [r_S(x_S, d_S) + f_{S-1}(t_S(x_S, d_S) + f_{M1}(t_{S1}(r_S, d_S))]$$

At this stage, the optimal return $f_{S+M1}(x_S)$ is the combination of the optimal return at stage S, $r_S(x_S, d_S)$ the optimal return from the main serial process preceding stage S, $f_{S-1}(t_S(x_S, d_{S1})$ and the optimal return from the branch, $f_{M1}(t_{S1}(x_S, d_S)$. For each possible value x_S, $f_{S+M1}(x_S)$ and d_S are reserved at this stage.

2.1.4) For the Remaining Stages (from stage S + 1 to stage N, the terminal node)

The optimal return at each remaining stage from S + 1 to N can be obtained as in the usual serial systems, i.e.,

$$f_{N+M1}(x_N) = \max_{1 \leq d_N \leq P_N} [r_N(x_N, d_N) + f_{N-1+M1}(t_N(x_N, d_N))]$$

where $m = S + 1, \ldots, N$

2.1.5) Determination of the Optimal Decision and Return at Each Stage

At the final stage N, the optimal input X_N^* to the system can be obtained by letting

$$f_{N+1}(x_N^*) = \max_{1 \leq x_N \leq K_N} \{f_{N+M1}(x_N)\}$$

With the optimal input x_N^* and optimal decision d_N^* obtained from a decision table, we can produce optimal stage return r_N^* and optimal stage output x_{N-1}^* as follows:

$$r_N^* = r_N(x_N^*, D_N^*)$$
$$x_{N-1}^* = t_N(x_N^*, D_N^*)$$

This process continues from stage N down to stage S + 1.

At junction stage (stage S), the optimal stage input x_S^* and stage decision d_S^* , the optimal branch input x_{M1}^* are obtained via the transition equation

$$x_{M1}^* = t_{S1}(x_S^*, d_S^*).$$

For the remaining processes, the stage transformation, return function, and decision tables can be used at each stage.

2.1.6) Input Data Required for the Algorithm

The input data for the algorithm are as follows:

N = # of stages in the main serial process
M = # of stages in the branch
S = junction stage
K_{m1} = upper bound in the input value x_{m1} , m = 1, 2, ..., M

2.1.7) Output List of the Algorithm

Basically, the output of this algorithm consists of the following:

1. Return table at each stage for each input value.

2. Decision table at each stage for each input value.

3. Optimal input, decision, and return at each stage.

In the next section, we present a companion flow chart for a computer program to implement the basic algorithm whose steps were detailed above. In Section 2.3, a simple problem is posed and solved. The situation involving a set of constraints is next treated.

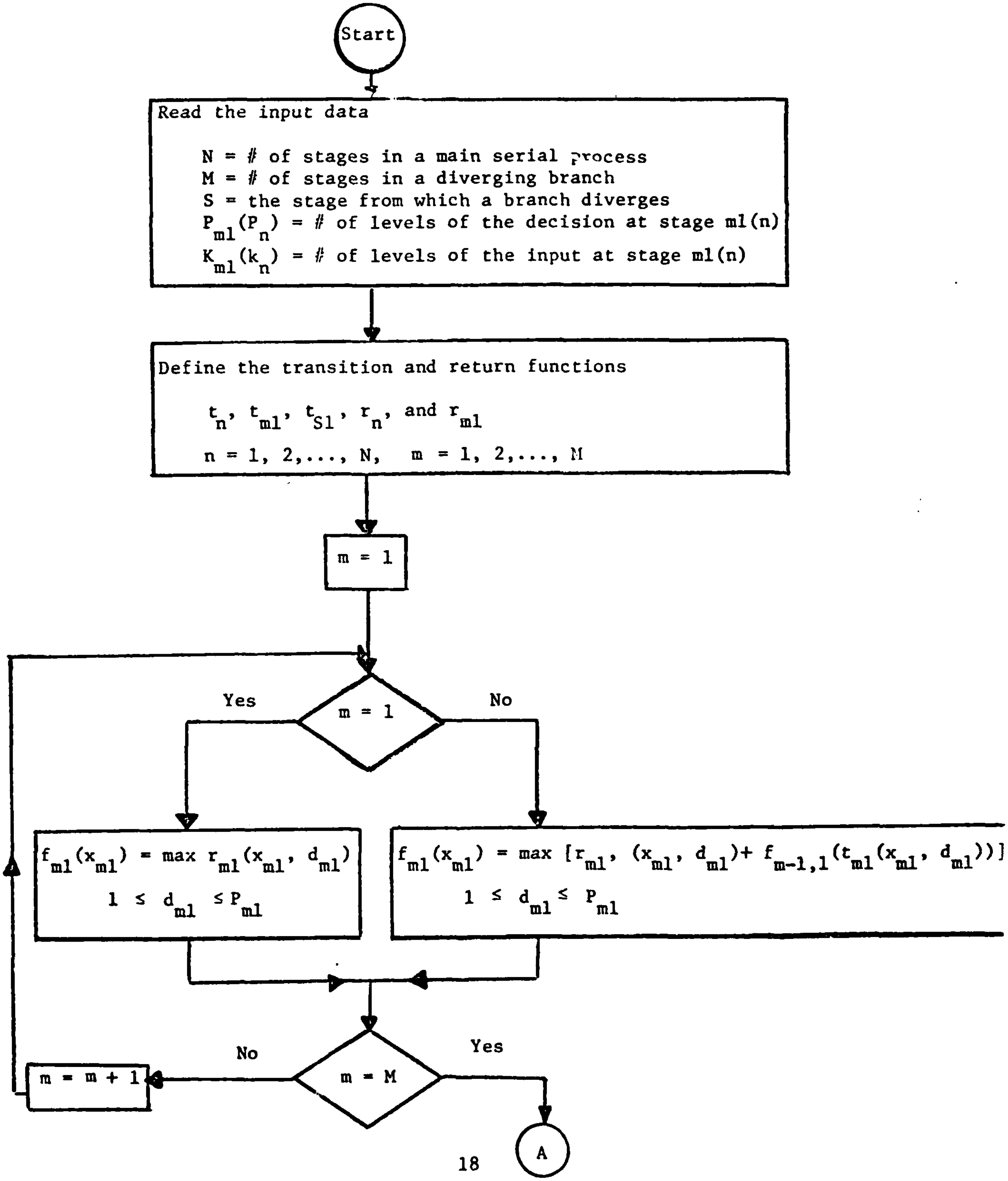
In Section 2.4, the diverging branch problem is analyzed in terms of the algorithm's sensitivity to a variation in N, the number of stages, the complexity of branches, as well as the transition functions.

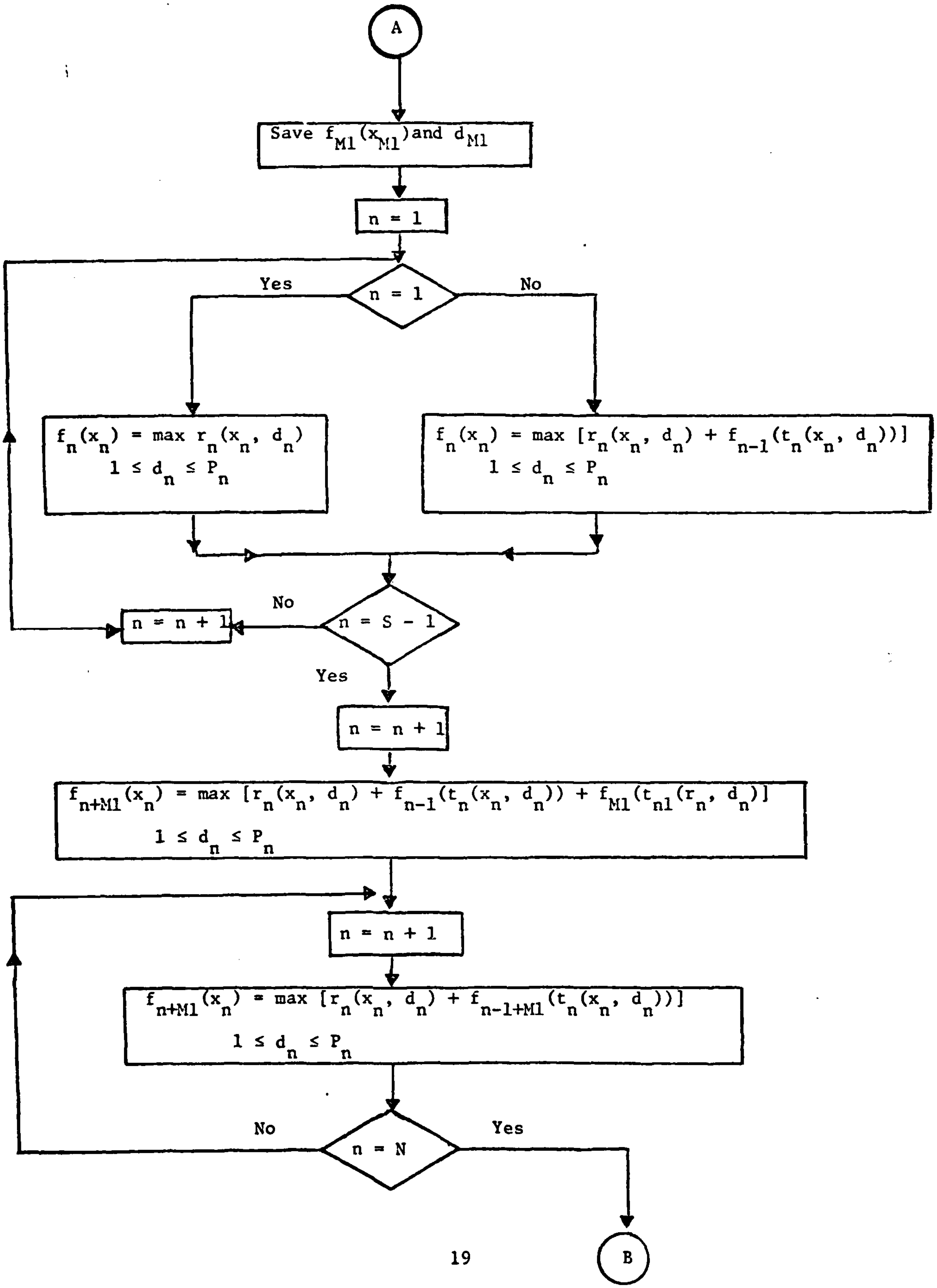
2.2 The Algorithm: Special Structure and Flowchart

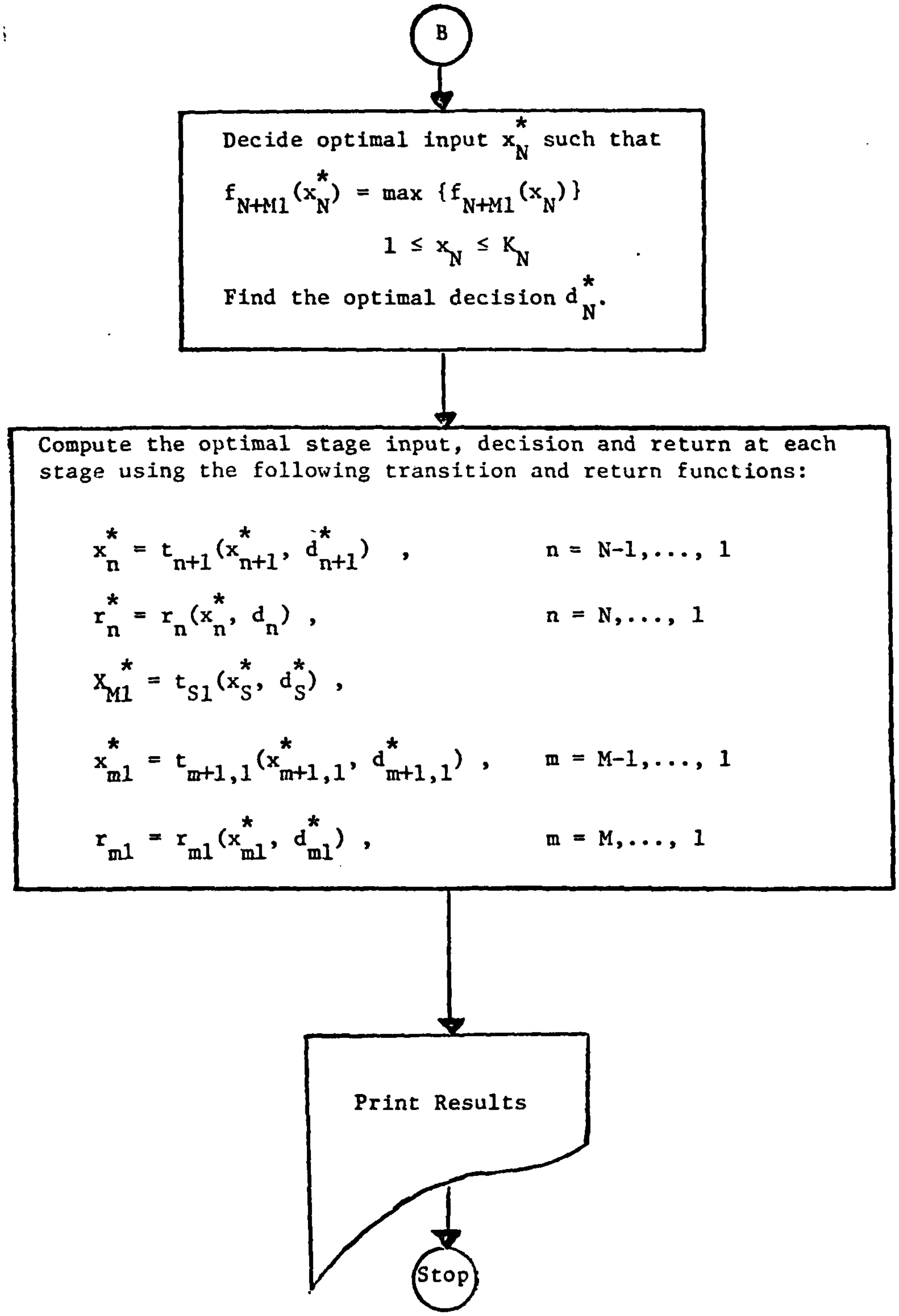
The high level algorithm developed above and flowcharted in the sequel, although akin to the conventional dynamic programming version, has some special structures worthy of note. A direct application of the usual approaches would dictate enormous storage requirements when processing nonserial networks, thus making the processing of large networks virtually impossible. To mitigate this problem, we devise a technique which enables us to indicate the optimal decision values at each stage by appending K_d = 1 + K to the state entry in the corresponding transaction table, where K = max $\{K_{11}, K_{12}, \ldots, K_{M1}; K_1, K_2, \ldots, K_N\}$. Although this adds one more state variable and this stage, it enables us to eliminate the storage requirement for the optimal decision which is normally the case with classical algorithms. Any future reference to this table entry is maie as the entry Mod K_d. When needed later, the optimal decision values can be retrieved by searching only one row of the table for the entry value greater than or equal to K_d.

The computer algorithms developed for the diverging branch system, as well as for other systems in this report, are done in Fortran. They are based on the analysis presented in the foregoing sections and flowcharted in Fig. 2.2. The input data dictated by the dynamic programming constructs are in terms of functions of the respective variables. This also makes it possible to handle larger problems without the forbidding storage limitations.

Fig. 2.2: Flow Chart of the Diverging Branch Algorithm


Start
Read the input data
N = # of stages in a main serial process
M = # of stages in a diverging branch
S = the stage from which a branch diverges
$P_{ml}(P_n)$ = # of levels of the decision at stage ml(n)
$K_{ml}(k_n)$ = # of levels of the input at stage ml(n)
Define the transition and return functions
t_n, t_{ml}, t_{S1}, r_n, and r_{ml}
n = 1, 2,..., N, m = 1, 2,..., M
m = 1
m = 1
Yes
No
$f_{ml}(x_{ml})$ = max $r_{ml}(x_{ml}, d_{ml})$
$1 \le d_{ml} \le P_{ml}$
$f_{ml}(x_{ml})$ = max $[r_{ml}, (x_{ml}, d_{ml}) + f_{m-1,1}(t_{ml}(x_{ml}, d_{ml}))]$
$1 \le d_{ml} \le P_{ml}$
m = M
No
Yes
m = m + 1
A


A
Save $f_{M1}(x_{M1})$ and d_{M1}
$n = 1$
Yes
$n = 1$
No
$f_n(x_n) = \max r_n(x_n, d_n)$
$1 \le d_n \le P_n$
$f_n(x_n) = \max [r_n(x_n, d_n) + f_{n-1}(t_n(x_n, d_n))]$
$1 \le d_n \le P_n$
No
$n = n + 1$
$n = S - 1$
Yes
$n = n + 1$
$f_{n+M1}(x_n) = \max [r_n(x_n, d_n) + f_{n-1}(t_n(x_n, d_n)) + f_{M1}(t_{n1}(r_n, d_n)]$
$1 \le d_n \le P_n$
$n = n + 1$
$f_{n+M1}(x_n) = \max [r_n(x_n, d_n) + f_{n-1+M1}(t_n(x_n, d_n))]$
$1 \le d_n \le P_n$
No
$n = N$
Yes
B



Decide optimal input x_N^* such that

$$f_{N+M1}(x_N^*) = \max_{1 \le x_N \le K_N} \{f_{N+M1}(x_N)\}$$

Find the optimal decision d_N^*.

Compute the optimal stage input, decision and return at each stage using the following transition and return functions:

$$x_n^* = t_{n+1}(x_{n+1}^*, d_{n+1}^*) \ , \qquad n = N-1, \ldots, 1$$

$$r_n^* = r_n(x_n^*, d_n) \ , \qquad n = N, \ldots, 1$$

$$X_{M1}^* = t_{S1}(x_S^*, d_S^*) \ ,$$

$$x_{m1}^* = t_{m+1,1}(x_{m+1,1}^*, d_{m+1,1}^*) \ , \qquad m = M-1, \ldots, 1$$

$$r_{m1} = r_{m1}(x_{m1}^*, d_{m1}^*) \ , \qquad m = M, \ldots, 1$$

2.3 Example (A Diverging Branch System)

Let us explicate the steps of this algorithm by considering an example. Suppose it is desired to maximize the sum of stage returns as a function of the input x_5 for the problem with the structure given in Fig. 2.3 and the data in Table 2.1.

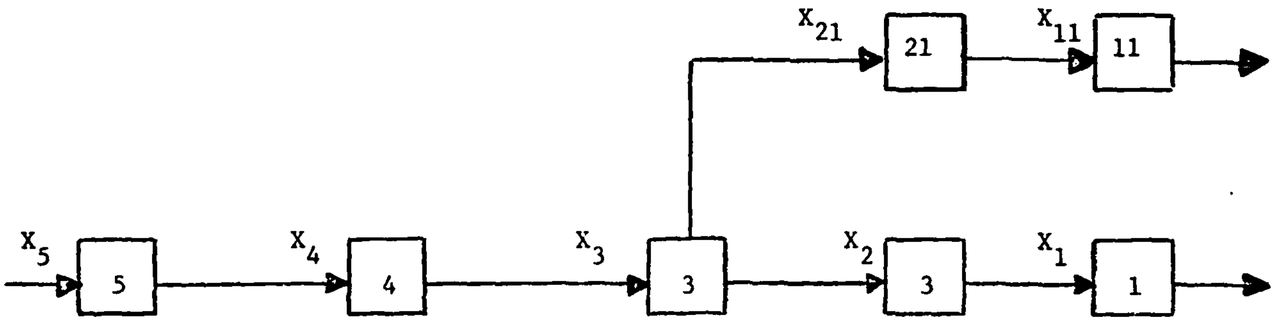


Figure 2.3: A Diverging Branch System Example

Table 2.1: The Return and the Transformation Function in the System.

Stage	Decision	Return	Transformation	Constraints
1	d_1	$r_1 = d_1^2$	-	$d_1 \varepsilon [\underline{d}_i, \bar{d}_i]$
2	d_2	$r_2 + d_2^2$	$x_1 = x_2 + d_2$	$x_i \varepsilon [\underline{x}_i, \bar{x}_i]$
3	d_3	$r_3 = d_3^2$	$x_2 = x_3 + d_3$	-
			$x_{21} = x_3 + d_3$	-
11	d_{11}	$r_{11} = d_{11}^2$	-	$d_{il} \varepsilon [\underline{d}_{il}, \bar{d}_{il}]$
21	d_{21}	$r_{21} = d_{11}^2$	$x_{11} = x_{21} + d_{21}$	$x_{il} \varepsilon [\underline{x}_{il}, \bar{x}_{il}]$
4	d_4	$r_4 = d_4^2$	$x_3 = x_4 + d_4$	-
5	d_5	$r_5 = d_5^2$	$x_4 = x_4 + x_5$	-

*$\underline{d}_i(\bar{d}_i)$, $\underline{x}_i(\bar{x}_i)$, $\underline{d}_{il}(\bar{d}_{il})$, and $\underline{x}_{il}(\bar{x}_{il})$ represent the lower bound (upper bound) of each corresponding variable.

In this system, the stages 11 to 21 and 1 to 2 can be optimized using usual recursive procedures as two disjoint serial systems.

At stage 11 and the diverging branch, we have

$$f_{11}(x_{11}) = \max_{d_{11}} d_{11}^2$$

and the optimal decision is found to be

$$d_{11}(x_{11}) = \bar{d}_{11}$$

Also, at stage 21, we see that

$$d_{21}(x_{21}) = \bar{d}_{21}$$

with $$f_{21}(x_{21}) = \bar{d}_{21}^2 + \bar{d}_{21}^2 .$$

For stages 1 and 2 of the main serial process, the optimal decision and the value of $f_2(x_2)$ are given by

$$d_1(x_1) = \bar{d}_1$$

$$d_2(x_2) = \bar{d}_2$$

with $$f_2(x_2) = \bar{d}_i^2 + \bar{d}_2^2.$$

Now, at stage 3 the two optimal returns $f_{21}(x_{21})$ and $f_2(x_2)$ arecombined with the stage 3 return as follows:

$$f_3(x_3) = \max_{d_3} [d_3^2 + (\bar{d}_{11}^2 + \bar{d}_{21}^2)+ (\bar{d}_1^2 + \bar{d}_2^2)]$$

We again observe that

$$d_3(x_3) = \bar{d}_3$$

with $$f_3(x_3) = \bar{d}_i^2 + \bar{d}_2^2 + \bar{d}_3^2 + \bar{d}_{11}^2 + \bar{d}_{21}^2$$

The optimization of the remaining stages, from stage 4 to 5, can also be carried out recursively.

We finally have the following optimal return function at stage 5.

$$f_5(x_5) = \sum_{i=1}^{5} \bar{d}_i^2 + \sum_{i=1}^{2} \bar{d}_{i1}^2$$

The optimal solution can be summarized as follows (Table 2.2)

Table 2.2: Optimal Solution to the System Posed in the Example

Stage	Optimal Input	Optimal Decision	Optimal Return
1	$x_1 = x_2 + \bar{d}_1$	$d_1 = \bar{d}_1$	$r_1 = \bar{d}_1^2$
2	$x_2 = x_3 + \bar{d}_3$	$d_2 = \bar{d}_2$	$r_2 = \bar{d}_2^2$
3	$x_3 = x_4 + \bar{d}_4$	$d_3 = \bar{d}_3$	$r_3 = \bar{d}_3^2$
11	$x_{11} = x_{21} + \bar{d}_5$	$d_{11} = \bar{d}_{11}$	$r_{11} = \bar{d}_{11}^2$
21	$x_{21} = x_3 + \bar{d}_3$	$d_{21} = \bar{d}_{21}$	$r_{21} = \bar{d}_{21}^2$
4	$x_4 = x_5 + \bar{d}_5$	$d_4 = \bar{d}_4$	$r_4 = \bar{d}_4^2$
5	x_5	$d_5 = \bar{d}_5$	$r_5 = \bar{d}_5^2$

We will now solve the diverging branch system defined in Table 2.2 and Table 2.1 using the computer algorithm given in Section 2.2.

Some constraints on the input and the decision variables are added as in the following problem.

$$\max \quad \sum_{n=1}^{5} r_n + \sum_{m=1}^{2} r_{m1}$$

$$\begin{aligned}
\text{s.t.} \quad & 1 \le d_n \le 3 && n = 1, \ldots, 5 \\
& 1 \le d_{m1} \le 3 && m = 1, 2 \\
& 1 \le x_n \le 20 && n = 2, \ldots, 5 \\
& 1 \le x_1 \le 5 && \\
& 1 \le x_{m1} \le 20 && m = 1, 2
\end{aligned}$$

The return and decision tables at each stage for each input are shown in the computer output attached (See Table 2.3). The optimal input, decision, and return at each stage are also shown in the output.

2.4 Analysis of the Diverging Branch System

In Section 2.1, we had assumed that the input and decision variables at each stage had the following integer values:

$$1 \leq x_{i1} \leq K_{i1} \qquad i = 1, 2, \ldots, M$$

$$1 \leq x_i \leq K_i \qquad i = 1, 2, \ldots, N$$

$$1 \leq d_{i1} \leq P_{i1} \qquad i = 1, 2, \ldots, M$$

$$1 \leq d_i \leq P_i \qquad i = 1, 2, \ldots, N$$

Let $K = \max \ (K_{11}, \ldots, K_{M1}; K_1, \ldots, K_N)$

Now, we will discuss the effects on the storage and the computer time of our algorithm.

First, let us define the storage requirement of the diverging branch algorithm as a function of K, as follows:

$$O(K) = (M + N + 3) K$$

The above storage requirement can be easily verified from the algorithm [16].

2.4.1) Sensitivity to N

Let us assume that the maximum value K remains unchanged with the increase of the number of stages N, in the main serial process. The storage requirement in this case increases by K with each additional increase in the number of N. Moreover, the requirement O(K) is approximately proportional to N if the number of stages in a diverging branch, M is relatively small.

Table 2.4 shows the computational results of the example in Section 2.3 with the increasing number of stages.

Table 2.3: Computer Output Showing Stage Returns and Decisions

```
*[illegible] TABLE AT EACH STAGE FOR EACH INPUT*

STAGE !   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20
-----------------------------------------------------------------------------------------------------------
 11   !   9.   9.   9.   9.   9.   9.   9    9.   9.   9.   9.   9.   9.   9.   9.   9.   9.   9.   9.   9.
 21   !  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  13.  10.   9.
  1   !   9.   9.   9.   9.   9.   9.   9    9.   9.   9.   9.   9.   9.   9.   9.   9.   9.   9    9.   9.
  2   !  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.  18.   0.   0.   9.
  3   !  45.  45.  45.  45.  45.  45.  45.  45.  45.  45.  45.  45.  45.  45.  40.  37.  27.  22.  19.   9.
  4   !  54.  54.  54.  54.  54.  54.  54.  54   54.  54.  54.  49.  46.  41.  38.  28.  23.  20.  10.   9.
  5   !  63.  63.  63.  63.  63.
-----------------------------------------------------------------------------------------------------------

*DECISION TABLE AT EACH STAGE FOR EACH INPUT*

STAGE !   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20
-----------------------------------------------------------------------------------------------------------
 11   !   3    3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   [illegible]   3.   3.
 21   !   3.   3.   3.   3.   3.   3.   3.   3.   3    3.   3.   3.   3    3.   3.   3.   3.   [illegible]   1.   3.
  1   !   3.   3.   3.   3.   3.   3    3.   3.   3.   3.   3.   3    [illegible]   3.   3.   3    3    [illegible]   3.   3.
  2   !   3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   [illegible]   3.   [illegible]   3.   3    [illegible]   0.   3.
  3   !   3.   3.   3.   3.   3.   3    3.   3.   3    3    3.   [illegible]   [illegible]   3.   [illegible]   [illegible]   [illegible]   [illegible]   1    [illegible]
  4   !   3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   3.   3    3.   [illegible]   1.   3.   [illegible]   [illegible]   1.   3.
  5   !   3.   3.   3.   3.   3.
-----------------------------------------------------------------------------------------------------------

------------------------------------------
*STAGE     INPUT     DECIS     RETURN*
------------------------------------------
   5          5       3.0       9.0
   4          8       3.0       9.0
   3         11       3.0       9.0
   2         14       3.0       9.0
   1         17       3.0       9.0
  21         14       3.0       9.0
  11         17       3.0       9.0
------------------------------------------

TOTAL OPTIMAL RETURN IS  63.00
```

Table 2.4: Computer Storage Requirements of the Diverging Branch Problem

Number of Stages N	Storage Requirement O(K)	CPU Time (Seconds)
5	200	.279
10	300	.371
15	400	.480

As shown in the above table, both the storage requirement and the CPU time seem to increase linearly with the increase of N.

2.4.2) Sensitivity to the Complexity of Branches

When the number of diverging branches increases, the storage requirement is not so simple as in the previous case. Consider a multi-diverging branch system in Fig. 2.4 where the number of branches is D and each branch has $M_i, i = 1, 2, \ldots, D$ stages. Whenever a diverging branch is added, the branch needs storage both for the branch return and for the optimal decision at each stage of the branch, hence, the storage requirement increases by $(M_i + 1)$ K, $i = 1, 2, \ldots, D$. Thus, the total storage requirement for the D different diverging branches can be represented as follows:

$$O(K) = \left(\sum_{i=1}^{D} (M_i + 1) + N + 2 \right) K.$$

2.4.3) Sensitivity to the Complexity of Transition Functions.

When the transition functions t_n, t_{ml}, and t_{Sl} can be represented as a simple linear combination of stage input x_n (or x_{ml}) and decision d_n (or d_{ml}), the storage requirement is not complex. However, if a system requires a transition function which has nonlinear term of x_n or d_n, the storage problem becomes serious.

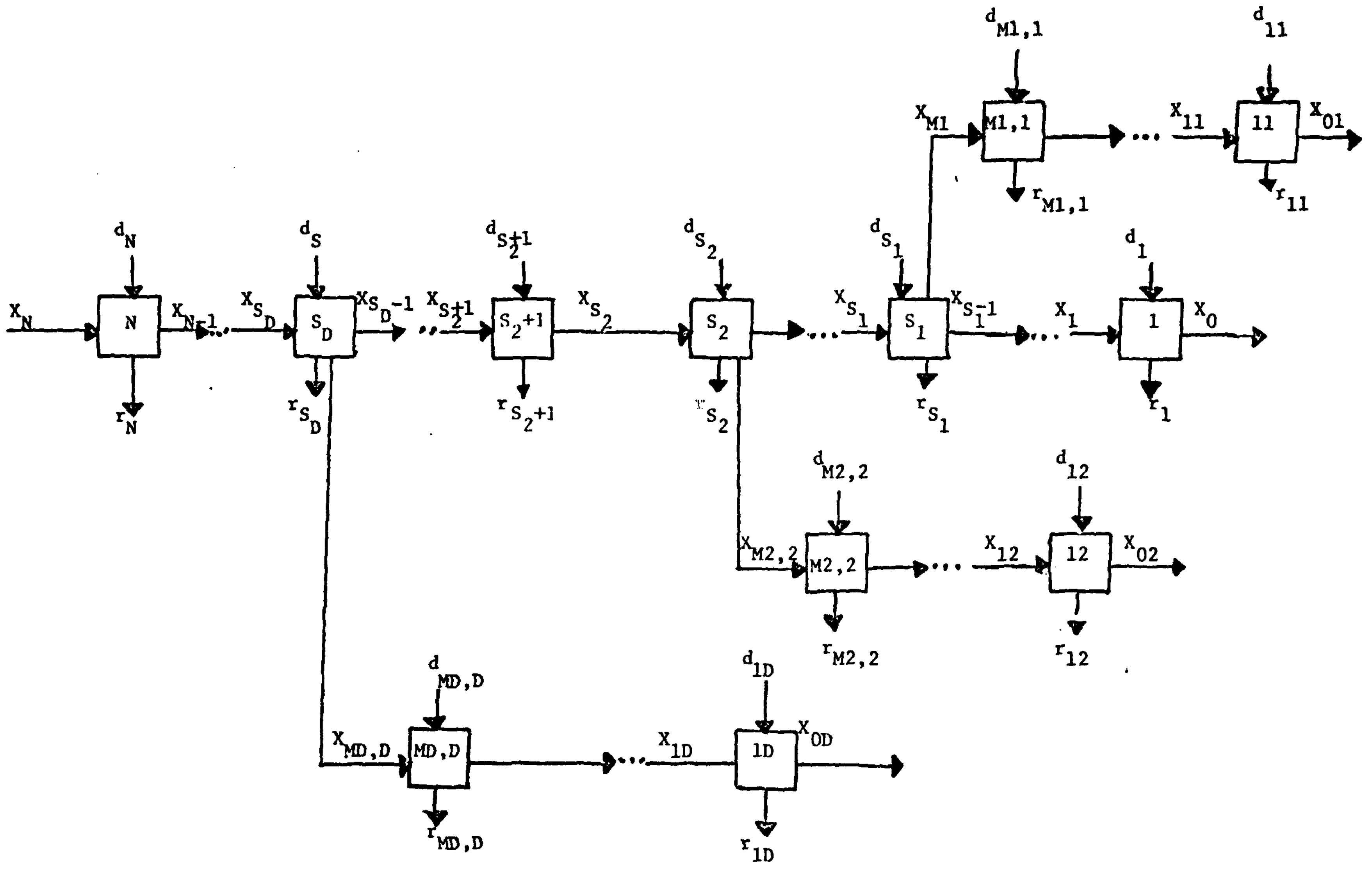


Figure 2.4: A Multi-Diverging Branch System

Consider the example in Section 2.3 if the transition function is given by

$$t_n(x_n, d_n) = x_n^2 + d_n \qquad \text{for } n = 1, 2, \ldots, 5.$$

then, $K = \max K_i$, $1 \leq i \leq 5$ becomes a huge number in an unconstrained problem. As a result, the computational time will be increased dramatically.

2.4.4) Sensitivity to the Complexity of Return Functions

Provided that we use a function of x_n and d_n for the return at each stage, the complexity of return function does not affect the storage requirement of this algorithm. This is because the sto.age requirement is strictly a function of K, N and M_i.

Chapter 3

ANALYSIS OF CONVERGING BRANCH NONSERIAL NETWORKS

3.1 Development of a DP Algorithm for the Converging Branch System

Let us direct our attention to the converse of the diverging branch system, namely a converging branch nonserial network. In its simplest form, a number of parallel serial systems join together at a junction node and then feed their outputs to a serial system. A simple example consisting of two input parallel branches and one serial output is exhibited in Fig. 3.1.

For analysis and algorithm development, consider this structure as a main serial system n, = 1, 2, ..., N and a branch m, = 1, 2, ..., M. The convergence occurs at node (stage) S. The transformation at this stage may be written as:

$$x_{S-1} = t_S(x_{01}, x_S, d_S)$$

The transition function for the other stages may be represented as in the usual serial processes as follows:

For the Branches

$$x_{m-1,1} = t_{m1}(x_{m1}, d_{m1}) \quad , \quad m = 1, 2, \ldots, M$$

For the Main

$$x_{n-1} = t_n(x_n, d_n) \quad , \quad \begin{matrix} n = 1, 2, \ldots, N \\ n \neq S \end{matrix}$$

We define the returns for each stage similarly. Thus,

$$r_S = r_S(x_{01}, x_S, d_S)$$

$$r_n = r_n(x_n, d_n) \quad , \; n = 1, 2, \ldots, N, \quad n \neq S$$

$$r_{m1} = r_{m1}(x_{m1}, d_{m1}) \quad , \; m = 1, 2, \ldots, M$$

To develop the algorithm, we proceed as follows. We first decompose the system into three components corresponding to stages 11, 21 to M1, and 1 to N. For stages 1 to N, we separately consider stages 1 to S - 1, S, and finally S + 1 to N. To find the optimal branch return $f_{M1}(x_{01})$ we will use the backward recursion. Next we will maximize $f_{M1}(x_{01})$ over x_{M1}. The recursion equations for the different phases may then be defined as follows: (Section 3.1.1)

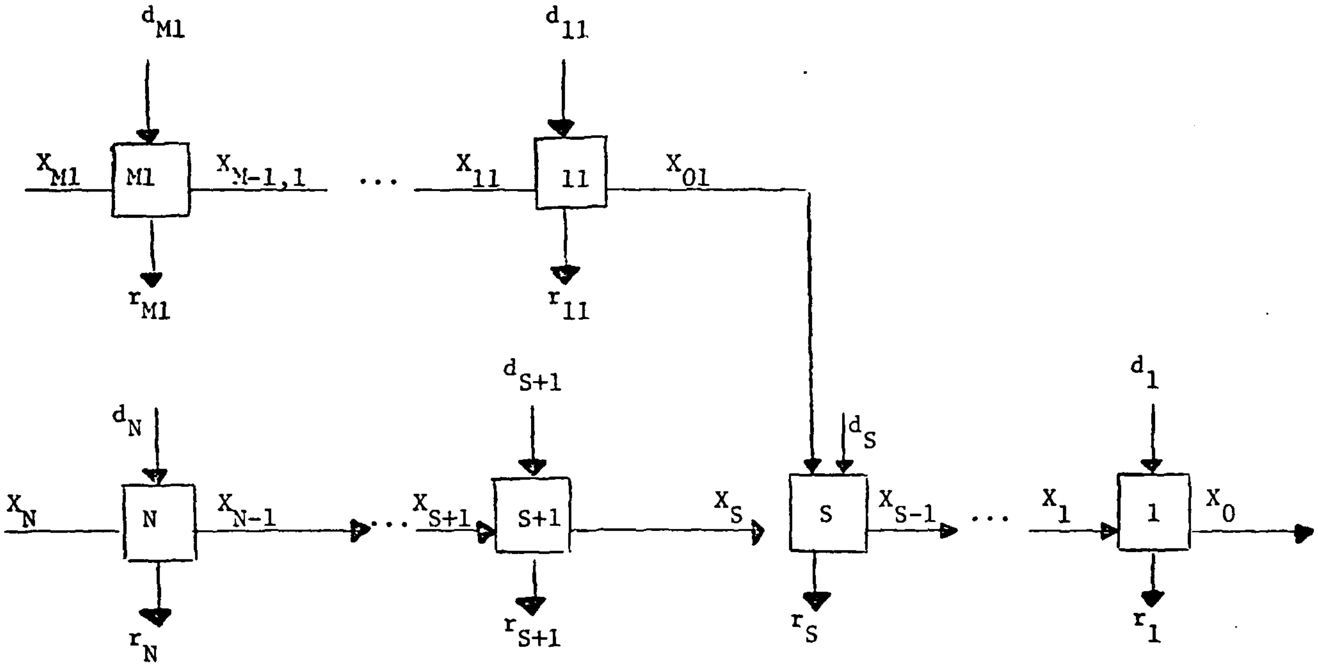


Figure 3.1: A Converging Branch System

3.1.1) For Stage 11

We solve the problem

$$f_{11}(x_{11}, x_{01}) = \underset{1 \quad d_{11} \quad P_{11}}{\max} r_{11}(x_{11}, d_{11})$$
$$\text{s.t.} \quad x_{01} = t_{11}(x_{11}, d_{11})$$

In other words, for each input value x_{11} we will find the optimal decision d_{11} which satisfies $x_{01} = t_{11}(x_{11}, d_{11})$ and also maximizes the stage return. For each value of (x_{11}, x_{01}), the optimal decision d_{11} and optimal return r_{11} are saved.

3.1.2) For Stages 21 to M1

The optimal return is given by

$$f_{m1}(x_{m1}, x_{01}) = \max_{1 \leq d_{m1} \leq P_{m1}} [r_{m1}(x_{m1}, d_{m1}) + f_{m-1,1}(t_{m1}(x_{m1}, d_{m1}))]$$

where $\quad m = 2, \ldots, M.$

At each stage from 21 to M1, the optimal decision d_{m1} and optimal return f_{m1} are computed for each pair of (x_{m1}, x_{01}). At stage M1, $f_{M1}(x_{M1}, x_{01})$ is found and the value of x_{M1} which maximizes the branch return for each value of x_{01} is obtained.

3.1.3) For the Main Serial Process

3.1.3-i) The optimal return from stages 1 to S - 1 can be found by using the usual recursive procedure, i.e.,

$$f_1(x_1) = \max_{1 \le d_1 \le P_1} r_1(x_1, d_1)$$

$$f_n(x_n) = \max_{1 \le d_n \le P_1} [r_n(x_n, d_n) + f_{n-1}(t_n(x_n, d_n))]$$

where $n = 2, 3, \ldots, S - 1$

3.1.3-ii) At Stage S (the junction node)

The optimal branch return $f_{M1}(x_{M1}, x_{01})$ is combined with the return at stage S and the optimal return from stages 1 through S - 1 using the recursion equation

$$f_S(x_S) = \max [r_S(x_{01}, x_S, d_S) + f_{S-1}(t_S(x_{01}, x_S, d_S)) + f_{M1}(x_{M1}, x_{01})]$$

where the maximization is over $1 \le x_{01} \le k_{01}$ and $1 \le d_S \le p_S$. In other words, at junction S, we compute the optimal return $f_S(x_S)$ and determine optimal branch output x_{01}, and optimal decision d_S, for each input value of x_S. We can also obtain the optimal branch input x_{M1} which maximizes the branch return using the value of x_{01}.

3.1.3-iii) For Stage S + 1 to N

The recursion equation is given by

$$f_N(x_N) = \max_{1 \le d \le P_N} [r_N(x_N, d_N) + f_{N-1}(t_N(x_N, d_N))]$$

where $n = S + 1, \ldots, N$

At the final stage N the optimal system return for each input value of x_n can be obtained.

3.1.4) Determination of the Optimal Decision and Return at Each Stage

At the final stage, the optimal input x_N^* can be obtained which maximizes $f_N(x_N)$ with the optimal decision d_N^* obtained from the decision table. We will proceed from stage N - 1 to stage S + 1. At stage S, the optimal input x_S^* and optimal branch input x_{01}^* are found as follows:

$$x_S^* = t_{S+1}(x_{S+1}, d_{S+1})$$

Now that x_S^* has been found, the optimal branch input x_{01}^* can be obtained. This is because we decided optimal x_{01} for each value of x_S when evaluating the optimal objective function value at stage S. For the remaining stages the optimal stage input and decision can be obtained using, the stage transformation function:

$$x_N = t_{N+1}(x_{N+1}, d_{N+1}), \; N = S - 1, S - 2, \ldots, 1$$

and the decision table, respectively.

3.1.5) Input Data Required for the Algorithm

The algorithm, akin to that developed for the diverging branch system, is designed to receive the following input specifications in Fortran:

N = # of stages in the main serial process

M = # of stages in the converging branch

S = junction stage

K_{i1} = upperbound of the input value x_{i1}, i = 1, 2, ..., M

K_i = upperbound of the input value x_i, i = 1, 2, ..., N

P_{i1} = upperbound of the decision value d_{i1}, i = 1, 2, ..., M

P_i = upperbound of the decision value d_i, 1, 2, ..., N

3.1.6) Output List of the Algorithm

At the completion of the operations, unless otherwise specified, the algorithm outputs the following:

1. Return and decision table for each pair of (x_{m1}, x_{01}), m = 1, 2, ..., M
2. Optimal branch input x_{01} and branch return for each value of x_{01}.
3. Decision table for main serial process.
4. Optimal branch output $x_{.1}$ for each value of junction input x_S.
5. Optimal input, decision and return at each stage.

3.2 The Algorithm, Flowchart, and Structure

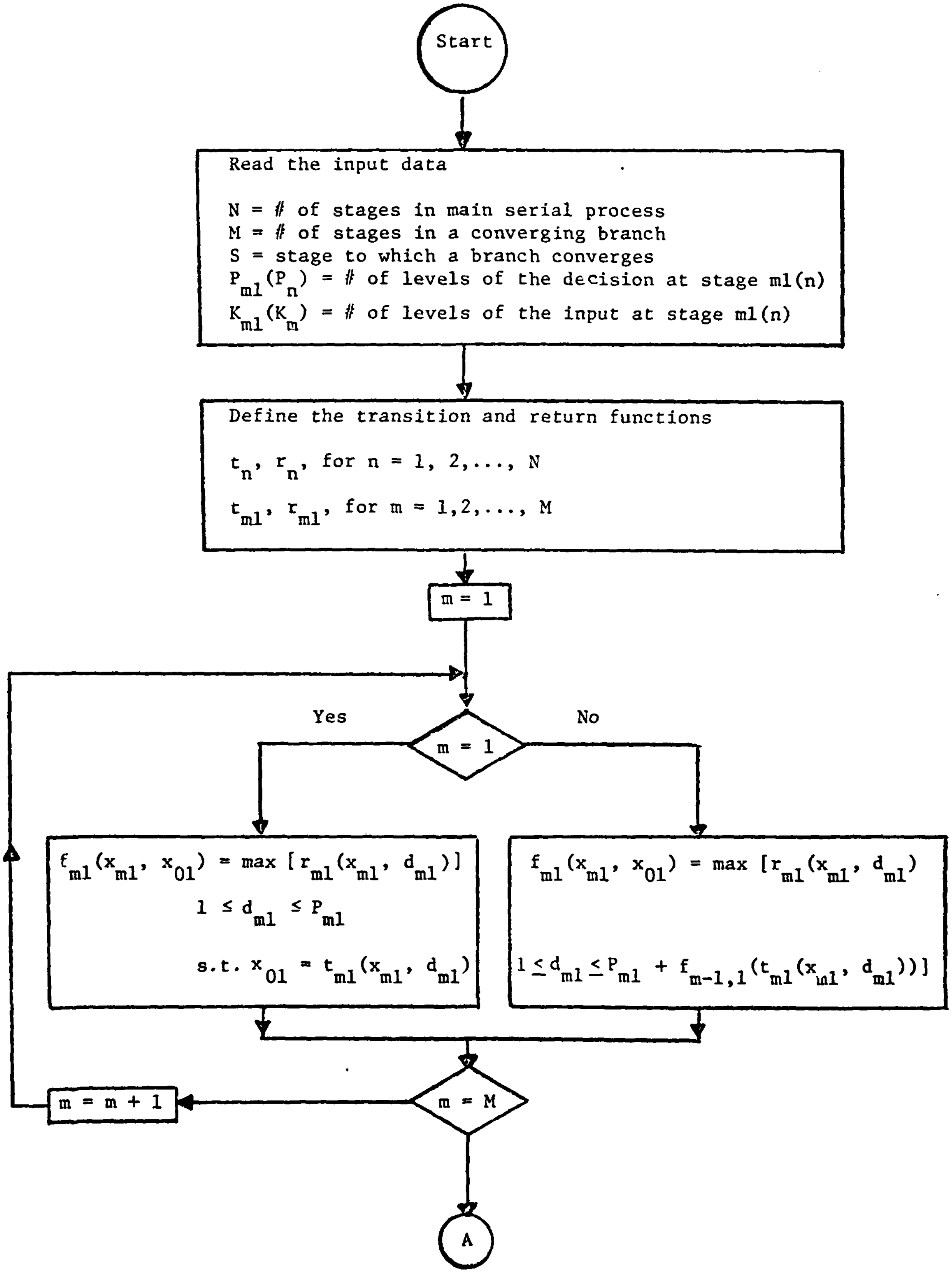
As is evident from the analysis of the foregoing section, the computational schema for the converging branch system differs from that developed for the diverging branch system. The flow chart for the converging branch system is given in Fig. 3.2. Although there are similarities in the logic of the flow charts, major differences occur in the optimization procedure as well as the difficulty of performing the optimization.

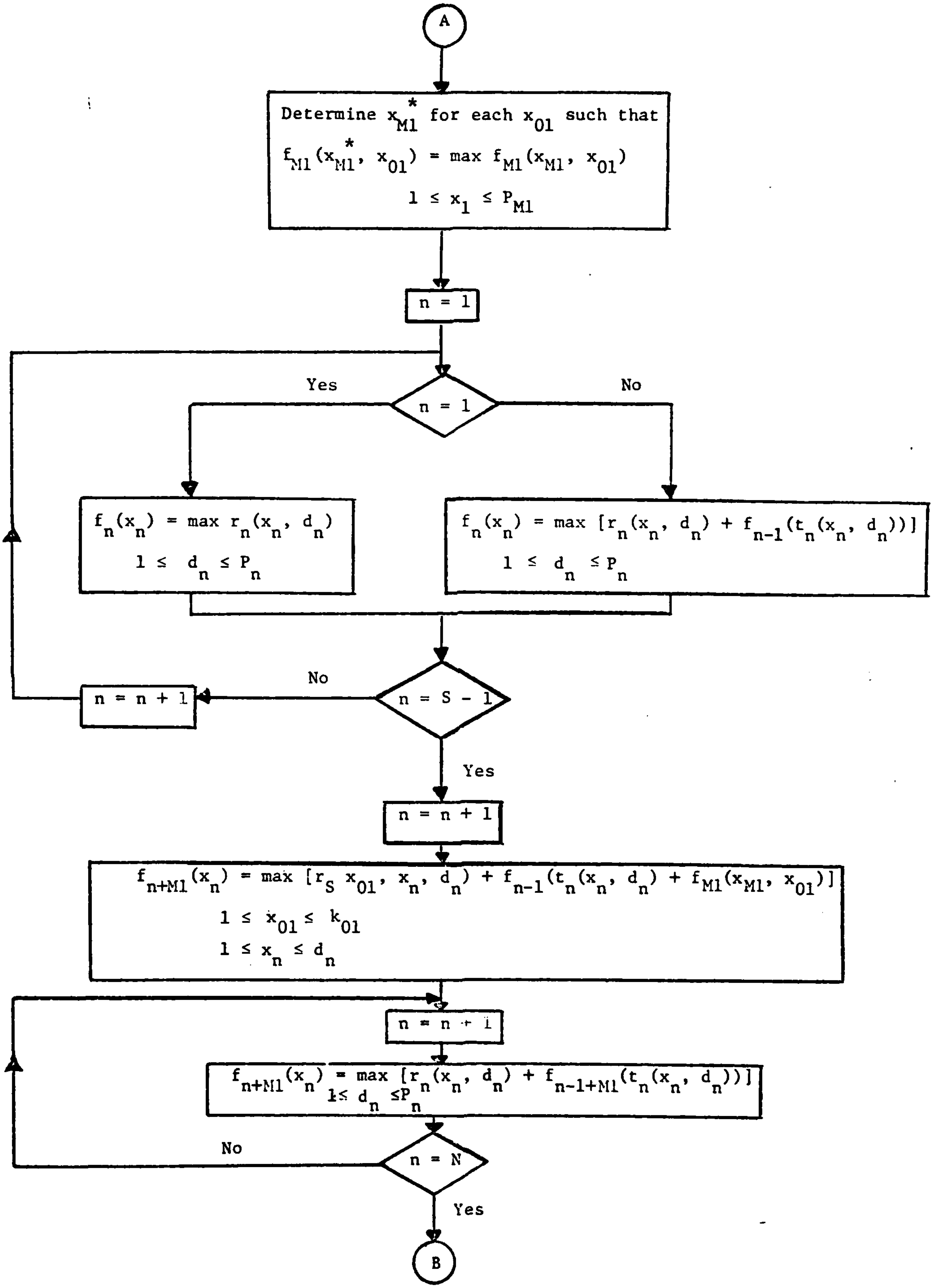
Comparing Figs. 2.2 and 3.2, we notice that the first major difference arises in section A where instead of storing the optimal return and decision at stage M1, a vector optimization involving two state variables x_{M1} and x_{01} is performed in the converging branch case. The differences surface again in the computation of $f_{n+M1}(x_n)$. From then on, the charts follow basically the same procedure.

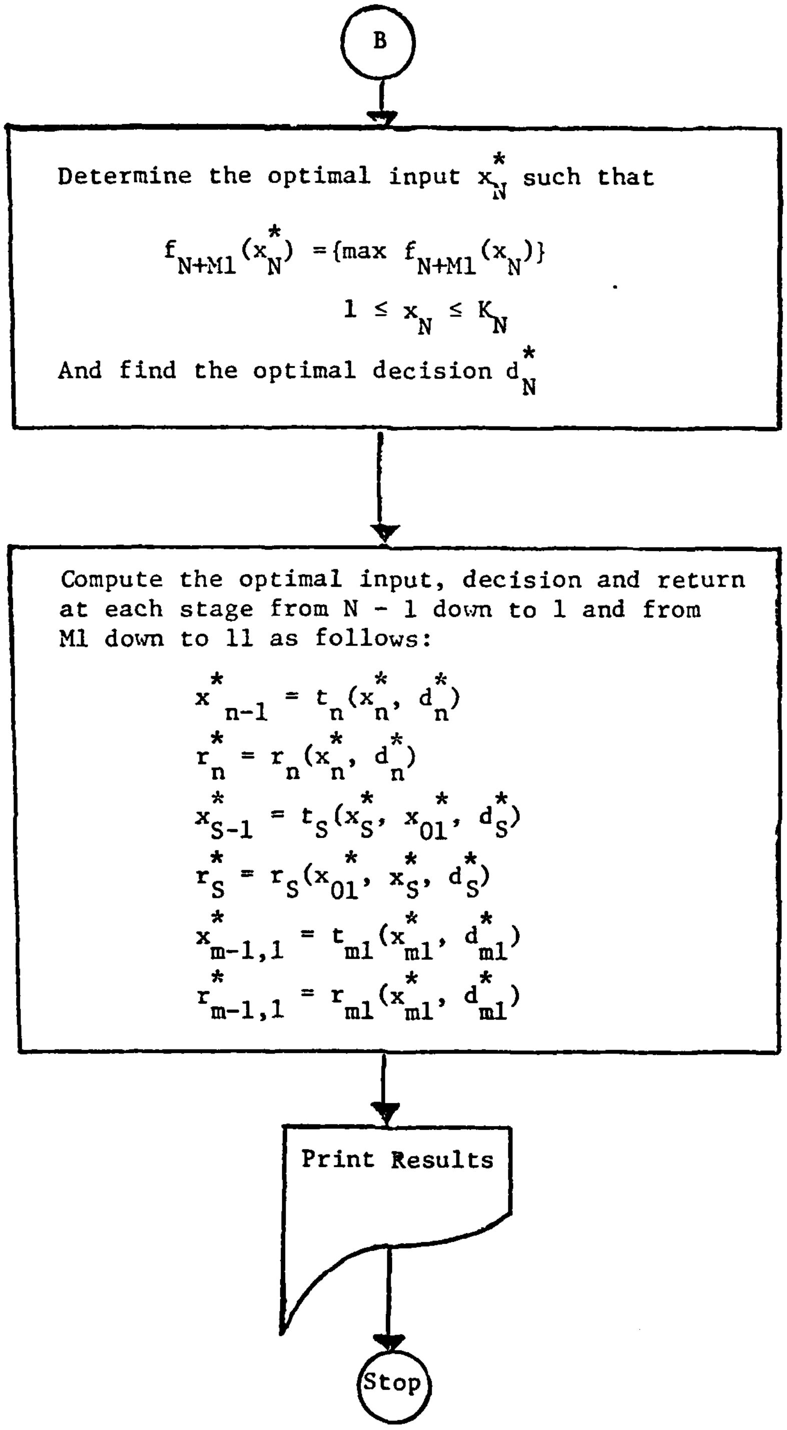
3.3 An Example Involving the Converging Branch Algorithm

Let us now demonstrate the use of the algorithm developed in section 3.1 in the solution of a converging system problem. Suppose it is desired to find the policy which maximizes the sum of the stage returns as a function of the inputs x_5 and x_{21} for the problem with the structure depicted in Fig. 3.3 using the data of Table 3.1. In this problem, the serial component consists of stages 1 through 5 while the converging branch consists of 11 and 21 with convergence occurring at node 3.

Fig. 3.2: Flow Chart of the Converging Branch Algorithm


Start
Read the input data
N = # of stages in main serial process
M = # of stages in a converging branch
S = stage to which a branch converges
$P_{m1}(P_n)$ = # of levels of the decision at stage m1(n)
$K_{m1}(K_m)$ = # of levels of the input at stage m1(n)
Define the transition and return functions
t_n, r_n, for n = 1, 2,..., N
t_{m1}, r_{m1}, for m = 1,2,..., M
m = 1
Yes
m = 1
No
$f_{m1}(x_{m1}, x_{01}) = \max_{1 \le d_{m1} \le P_{m1}} [r_{m1}(x_{m1}, d_{m1})]$
s.t. $x_{01} = t_{m1}(x_{m1}, d_{m1})$
$f_{m1}(x_{m1}, x_{01}) = \max_{1 \le d_{m1} \le P_{m1}} [r_{m1}(x_{m1}, d_{m1}) + f_{m-1,1}(t_{m1}(x_{m1}, d_{m1}))]$
m = m + 1
m = M
A


A
Determine x^*_{M1} for each x_{01} such that
$f_{M1}(x^*_{M1}, x_{01}) = \max f_{M1}(x_{M1}, x_{01})$
$1 \le x_1 \le P_{M1}$
n = 1
Yes
No
n = 1
$f_n(x_n) = \max r_n(x_n, d_n)$
$1 \le d_n \le P_n$
$f_n(x_n) = \max [r_n(x_n, d_n) + f_{n-1}(t_n(x_n, d_n))]$
$1 \le d_n \le P_n$
No
n = n + 1
n = S - 1
Yes
n = n + 1
$f_{n+M1}(x_n) = \max [r_S\ x_{01}, x_n, d_n) + f_{n-1}(t_n(x_n, d_n) + f_{M1}(x_{M1}, x_{01})]$
$1 \le x_{01} \le k_{01}$
$1 \le x_n \le d_n$
n = n + 1
$f_{n+M1}(x_n) = \max [r_n(x_n, d_n) + f_{n-1+M1}(t_n(x_n, d_n))]$
$1 \le d_n \le P_n$
No
n = N
Yes
B


B
Determine the optimal input x^*_N such that
$f_{N+M1}(x^*_N) = \{\max f_{N+M1}(x_N)\}$
$1 \le x_N \le K_N$
And find the optimal decision d^*_N
Compute the optimal input, decision and return at each stage from N - 1 down to 1 and from M1 down to 11 as follows:
$x^*_{n-1} = t_n(x^*_n, d^*_n)$
$r^*_n = r_n(x^*_n, d^*_n)$
$x^*_{S-1} = t_S(x^*_S, x^*_{01}, d^*_S)$
$r^*_S = r_S(x^*_{01}, x^*_S, d^*_S)$
$x^*_{m-1,1} = t_{m1}(x^*_{m1}, d^*_{m1})$
$r^*_{m-1,1} = r_{m1}(x^*_{m1}, d^*_{m1})$
Print Results
Stop

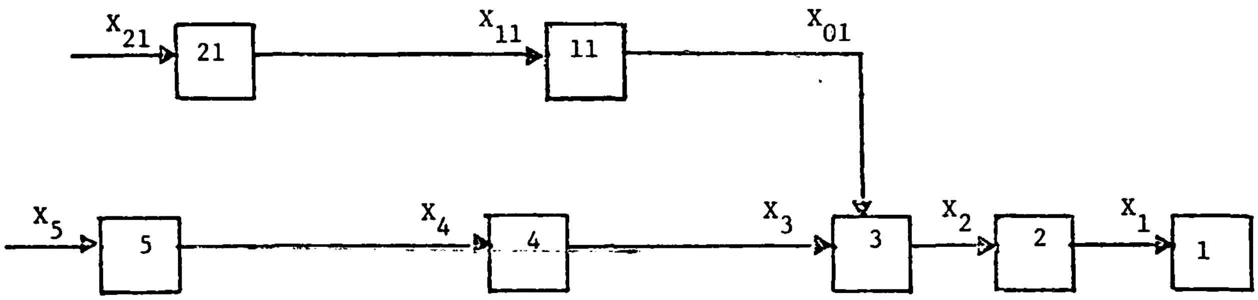


Figure 3.3: A Converging Branch System Example

Table 3.1: The Return and Transformation Function in the System

Stage	Decision	Return	Transformation	Constraints
1	d_1	$r_1 = d_1^2$	-	
2	d_2	$r_2 = d_2^2$	$x_1 = x_2 + d_2$	$d_i \varepsilon [\underline{d}_i, \bar{d}_i]$
3	d_3	$r_3 = d_3^2$	$x_2 = x_{01} + x_3 + d_3$	$x_i \varepsilon [\underline{x}_i, \bar{x}_i]$
11	d_{11}	$r_{11} = d_{11}^2$	$x_{01} = x_{11} + d_{11}$	
21	d_{21}	$r_{21} = d_{21}^2$	$x_{11} = x_{21} + d_{21}$	$d_{i1} \varepsilon [\underline{d}_{i1}, \bar{d}_{i1}]$
4	d_4	$r_4 = d_4^2$	$x_3 = x_4 + d_4$	$x_{i1} \varepsilon [\underline{x}_{i1}, \bar{x}_{i1}]$
5	d_5	$r_5 = d_5^2$	$x_4 = x_5 + d_5$	

*$\underline{d}_i$, $\underline{x}_i$, $\underline{d}_{i1}$ and $\underline{x}_{i1}$ represent the lowerbound of each variable and $\bar{d}_i$, $\bar{x}_i$, $\bar{d}_{i1}$, and $\bar{x}_{i1}$ represent the upperbound.

We will first begin with the branch system and determine $f_{21}(x_{21}, x_{01})$.

At stage 11,

$$f_{11}(x_{11}, x_{01}) = \max_{d_{11}} d_{11}^2$$

$$\text{s.t.} \quad x_{01} = x_{11} + c_{11}$$

Substituting the constraint into the recursion equation, we have

$$f_{11}(x_{11}, x_{01}) = (x_{01} - x_{11})^2$$

At stage 2, the recursion equation is given by

$$f_{21}(x_{21}, x_{01}) = \max_{d_{21}} [d_{21}^2 + (x_{01} - x_{21} - d_{21})^2]$$

Since $f_{21}(x_{21}, x_{01})$ is a convex function of the decision variable d_{21}, it is easy to see that the optimal decision variable with the value of $f_{21}(x_{21}, x_{01})$ is given by

$$d_{21}(x_{21}, x_{01}) = \bar{d}_{21}$$

$$d_{21}(x_{21}, x_{01}) = 2\bar{d}_{21}^2 + 2(x_{21} - x_{01})\,\bar{d}_{21} + (x_{21} - x_{01})^2$$

Now, the main serial system through stage 2 is optimized in the usual way to obtain $f_2(x_2)$. At stage 1, we have

$$f_1(x_1) = \max d_1^2$$

and the optimal decision is given by

$$d_1(x_1) = \bar{d}_1.$$

By proceeding as at stage 1, we have

$$d_2(x_2) = \bar{d}_2$$

$$f_2(x_2) = \bar{d}_1^{\,2} + \bar{d}_2^{\,2}$$

At stage 3, the branch return is combined with the serial return from stage 1 to stage 2 using the recursion equation

$$f_{3+21}(x_3, x_{21}) = \max [d_3^2 + (\bar{d}_1^{\,2} + d_2^2) + \{2\bar{d}_{21}^2 + 2(x_{21} - x_{01})\,\bar{d}_{21}$$
$$+ (x_{21} - x_{01})^2\}]$$

where the maximization is taken over x_{01} and d_3.

The optimal value of x_{01} is found to be

$$x_{01}(x_3, x_{21}) = \bar{x}_{01}$$

with $$d_3(x_3, x_{21}) = \bar{d}_3$$

and $$f_{3+21}(x_3, x_{21}) = \bar{d}_1^{\,2} + \bar{d}_2^{\,2} + \bar{d}_3^{\,2} + 2\bar{d}_{21}^2 + 2(x_{21} - \bar{x}_{01})\bar{d}_{21} + (x_{21} - \bar{x}_{01})^2 .$$

Having combined the branch optimally with the main serial systems we accomplish the analysis of stage 4 and 5 with the standard backward recursion. Then we finally have

$$d_4(x_4) = \bar{d}_4$$

$$d_5(x_5) = \bar{d}_5$$

$$f_{5+21}(x_5, x_{21}) = \sum_{i=1}^{5} \bar{d}_i^{\,2} + 2\bar{d}_{21}^{\,2} + 2(x_{21} - \bar{x}_{01})\bar{d}_{21} + (x_{21} - \bar{x}_{01})^2$$

Retracing our analysis, we find the remainder of the optimal solution, which is summarized in Table 3.2 as follows:

Table 3.2: Optimal Solution to the System

Stage	Optimal Input	Optimal Decision	Optimal Return
1	$x_1 = x_5 + x_{21} + \bar{d}_2 + \bar{d}_3 + \bar{d}_4 + \bar{d}_5 + \bar{d}_{11} + \bar{d}_{21}$	$d_1 = \bar{d}_1$	$r_1 = \bar{d}_1^{\,2}$
2	$x_2 = x_5 + x_{21} + \bar{d}_3 + \bar{d}_4 + \bar{d}_5 + \bar{d}_{11} + \bar{d}_{21}$	$d_2 = \bar{d}_2$	$r_2 = \bar{d}_2^{\,2}$
3	$x_3 = x_5 + \bar{d}_4 + \bar{d}_5$	$d_3 = \bar{d}_3$	$r_3 = \bar{d}_3^{\,2}$
	$x_{01} = x_{21} + \bar{d}_{21} + \bar{d}_{11}$	-	-
11	$x_{11} = x_{21} + \bar{d}_4$	$d_{11} = \bar{d}_{11}$	$r_{11} = \bar{d}_{11}^2$
21	x_{21}	$d_{21} = \bar{d}_{21}$	$r_{21} = \bar{d}_{21}^2$
4	$x_4 = x_5 + \bar{d}_5$	$d_4 = \bar{d}_4$	$r_4 = \bar{d}_4^{\,2}$
5	x_5	$d_5 = \bar{d}_5$	$r_5 = \bar{d}_5^{\,2}$

Let us now solve the converging branch system defined in Fig. 3.3 and Table 3.1 using the computer algorithm given in Section 3.2. We will add some constraints on the variables as in the following problem:

$$\max \sum_{n=1}^{5} r_n + \sum_{m=1}^{2} r_{m1}$$

$$\begin{aligned} \text{s.t. } & 1 \le d_n \le 3 && n = 1, 2, \ldots, 5 \\ & 1 \le d_{m1} \le 3 && m = 1, 2 \\ & 1 \le x_n \le 30 && n = 1, \ldots, 4 \\ & 1 \le x_1 \le 5 && \\ & 1 \le x_{m1} \le 10 && m = 0, 1, 2 \end{aligned}$$

The computational results are shown in the computer output of Table 3.3. The last table shows the optimal decision and return at each stage.

3.4 Analysis of the Converging Branch System

Different from the diverging branch system the storage requirement of the converging branch system depends not only on the value K but on K_{01} and K_S. these are defined as follows

$$K = \max (K_{11}, K_{21}, \ldots, K_{M1}, K_1, \ldots, K_N)$$

$$1 \le x_{01} \le K_{01}$$

and

$$1 \le x_S \le K_S$$

The storage requirement for the process of main serial is (N + 2)K , (N x K) for the decision value and (2 x K) for the stage return as discussed in the diverging branch system. See Section 2.4.

```
"DECISION TABLE AT STAGE 11"

  25.  24.  23.  22.  21.  20.  19.  18.  17.  16.  15.  14.  13.  12.  11.*************************
  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.  16.  15.  14.  13.  12.  11.********************
  27.  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.  16.  15.  14.  13.  12.  11.***************
  28.  27.  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.  16.  15.  14.  13.  12.  11.**********
  29.  28.  27.  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.  16.  15.  14.  13.  12.  11.*****
  30.  29.  28.  27.  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.  16.  15.  14.  13.  12.  11.
*****  30.  29.  28.  27.  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.  16.  15.  14.  13.  12.
**********  30.  29.  28.  27.  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.  16.  15.  14.  13.
***************  30.  29.  28.  27.  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.  16.  15.  14.
********************  30.  29.  28.  27.  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.  16.  15.
*************************  30.  29.  28.  27.  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.  16.
******************************  30.  29.  28.  27.  26.  25.  24.  23.  22.  21.  20.  19.  18.  17.
***********************************  30.  29.  28.  27.  26.  25.  24.  23.  22.  21.  20.  19.  18.
****************************************  30.  29.  28.  27.  26.  25.  24.  23.  22.  21.  20.  19.
*********************************************  30.  29.  28.  27.  26.  25.  24.  23.  22.  21.  20.
**************************************************  30.  29.  28.  27.  26.  25.  24.  23.  22.  21.
*******************************************************  30.  29.  28.  27.  26.  25.  24.  23.  22.
************************************************************  30.  29.  28.  27.  26.  25.  24.  23.
*****************************************************************  30.  29.  28.  27.  26.  25.  24.
**********************************************************************  30.  29.  28.  27.  26.  25.

DECISION TABLE AT STAGE 21

-439.-438.-437.-436.-435.-434.-433.-432.-431.-430.-429.-428.-427.-426.-425.-424.-423.-422.-421.-420.
-439.-439.-438.-437.-436.-435.-434.-433.-432.-431.-430.-429.-428.-427.-426.-425.-424.-423.-422.-421.
-439.-439.-439.-438.-437.-436.-435.-434.-433.-432.-431.-430.-429.-428.-427.-426.-425.-424.-423.-422.
-439.-439.-439.-439.-438.-437.-436.-435.-434.-433.-432.-431.-430.-429.-428.-427.-426.-425.-424.-423.
-439.-439.-439.-439.-439.-438.-437.-436.-435.-434.-433.-432.-431.-430.-429.-428.-427.-426.-425.-424.
-439.-439.-439.-439.-439.-439.-438.-437.-436.-435.-434.-433.-432.-431.-430.-429.-428.-427.-426.-425.
```

Table 3.3: Computer Output of Example of Fig. 3.3

```
"OPTIMAL BRANCH INPUT & RETURN FOR EACH X01"

 " X01  INPUT  RETURN"
-464        0.     -4890.
-463        0.     -4880.
-462        0.     -4870.
-461        0.     -4860.
-460        0.     -4850.
-459        0.     -4840.
-458        0.     -4830.
-457        0.     -4820.
-456        0.     -4810.
-455        0.     -4800.
-454        0.     -4790.
-453        0.     -4780.
-452        0.     -4770.
-451        0.     -4760.
-450        0.     -4750.
-449        0.     -4740.
-448        0.     -4730.
-447        0.     -4720.
-446        0.     -4710.
-445        0.     -4700.

 "DECISION TABLE FOR MAIN PROCESS"

  46.  47.  48.  49.  50.  50.  50.  50.  50.  50.  50.  50.  50.  50.  50.  50.  50.  50.  50.  50.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 500. 500. 500. 500. 500. 500.

 "OPTIMAL INPUT X01 FROM BRANCH TO JUNCTION"

-445.-445.-445.-445.-445.-446.-447.-448.-449.-450.-451.-452.-453.-454.-455.-456.-457.-458.-459.-460.

 "STAGE  INPUT  DECIS  RETURN"

    3      5    500  -20250.
    2    505      0   49900.
    1    505     50    -250.
    1   -455
    2      0   -430   -4675.          TOTAL OPTIMAL RETURN IS 24600.
    1   -430     25    -125.                  1.177 CP SECONDS EXECUTION TIME.
```

Table 3.3 Continued

However, the storage problem for the converging branch is a little complicated. It can be analyzed as in the following cases.

1. At each stage (11, 21,..., M1) for each pair of input to the stage and the branch output (x_{01}), we determine the decision value which optimizes the branch return. Hence, the algorithm needs 2 X K X K_{01} storage for computing the total branch return for each pair of branch input x_{M1} and branch output x_{01}. Another M X K X K_{01} storage is required for the decision value at each stage. Therefore, it becomes (M + 2) X K X K_{01}.
2. Storage requirements for the decision of optimum branch input and branch return for each value of branch output x_{01} becomes 2 X K_{01}.
3. At junction stage S, we need to determine the optimal value of the input from the converging branch for each possible value of the input from stage S +1 to S. This requirement becomes K_S.

From the above analysis, the total storage requirement of the converging branch algorithm can be represented as a function of K, K_{01} and K_S as follows:

$$O(K, K_{01}, K_S) = (N + 2)K + (M + 2) X K X K_{01} + 2K_{01} + K_S$$

If we assume the levels of the state variables x_{01} and x_S to be equal to K, then the above expression becomes

$$O(K) = (M + 2) K^2 + (N + 5)K$$

Next, we will discuss the effects of storage, and time for the following cases.

3.4.1) Sensitivity to N

As shown in the function O(K), the storage requirement increases by K as we increase one stage of the main serial process. This is the same as in the diverging branch system.

However, notice that the storage requirement in the converging branch is highly affected by the number of stages in the branch. This is because each additional increase of M requires a storage of the order K^2.

The computational results with increasing number of N in the example in Section 3.3 are summarized in the following table:

Table 3.4: Computational Experience with the Converging Branch System

Number of Stages N	Storage Requirements O(K)	CPU Time (Seconds)
5	1440	.496
10	1590	.695
15	1740	.841

3.4.2) Sensitivity to the Complexity of Branches

As discussed before, the storage requirement in a converging branch system is highly affected by the number of stages in a converging branch. Moreover, if the number of branches increases, the problem becomes really serious.

Consider the multi-converging branch system shown in Fig. 3.4, where the number of converging branches is D and each branch has M_i, $i = 1, 2, \ldots, D$ stages in it. When a converging branch is added, the branch needs storage of the order of $(M_i + 2)K^2 + 3K$ as analyzed before.

Hence, the storage requirement for the above multi-converging branch system can be $O(K) = (\sum_{i=1}^{D} M_i + 2)K^2 + (N + 2 + 3D)K$.

3.4.3) Sensitivity to the Complexity of Transition Functions

As we have discussed in the diverging branch system, the complexity of transition function affects the levels of discretization of the input state at each stage. Hence, it directly affects the maximum number K on which the

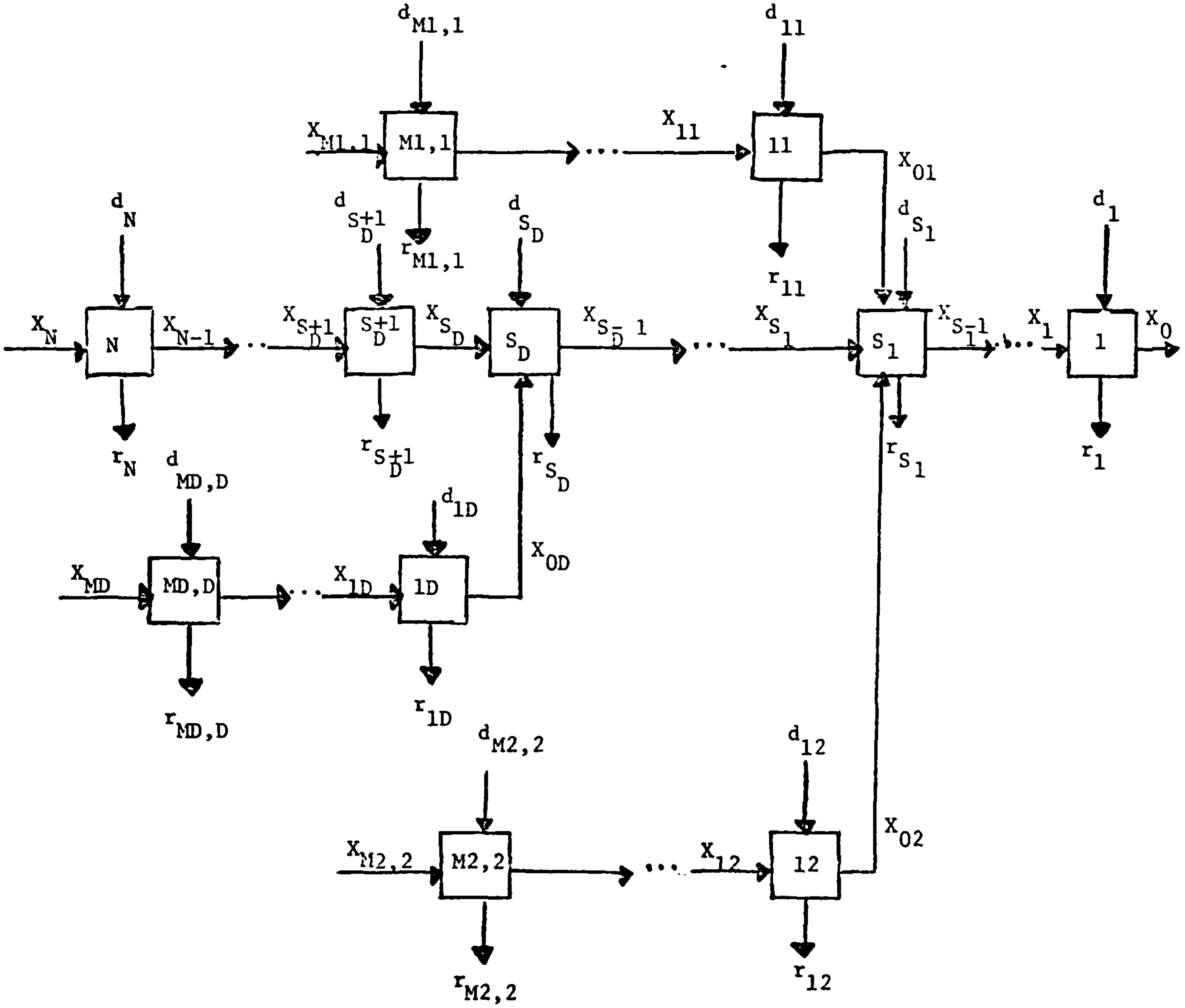


Figure 3.4: A Multi-Converging Branch System.

storage requirements mainly depends.

3.4.4) Sensitivity to the Complexity of Return Functions

Again, the return function has no impact on the storage requirement and computational time providing the algorithm employs a function in representing the input variables x_n and decision d_n.

3.5 A Modified Converging Branch Algorithm

The algorithm developed in Section 3.1 and used to solve the sample problem in Section 3.3 is somewhat restricted in the classes of problems it solves. To use it in solving problems involving complex functions, especially with regards to the transition and return functions, a modification was necessary. This led to the development of the modified algorithm. Let us then illustrate the use of the modified converging branch algorithm with the following example problem. Suppose it is desired to find the policy which minimizes the sum of the stage returns as a function of the inputs x_3 and x_{21}, for the problem with the structure given in Fig. 3.5 using the data given in Table 3.5. First notice the presence of fractional numbers in the return functions for each stage. Since there are no stages after the converging stage in the main serial system, we begin with the two stage branch system, and determine $f_{21}(x_{21}, x_{01})$. At stage 11

$$f_{11}(x_{11}, x_{01}) = \min_{d_{11}} 0.2\, d_{11}^2$$

$$\text{s.t. } x_{01} = x_1 = d_{11}$$

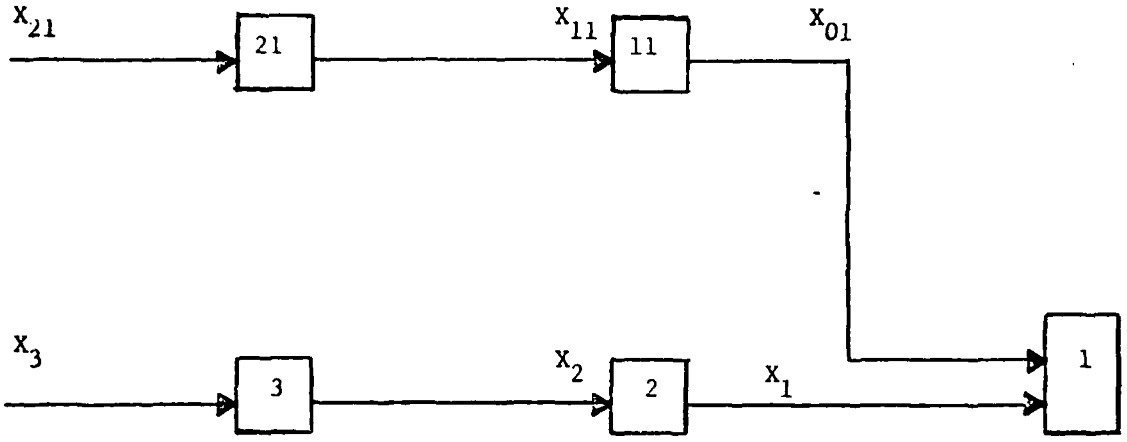


Figure 3.5 : A Converging Branch System Example

Table 3.5 The Return and Transformation Functions with Constraints in the System

Stage	Decision	Return	Transformation	Constraint
1	d_1	$r_1 = 0.1\ d_1^2$	-	$d_1 = x_1 + x_{01}$
11	d_{11}	$r_{11} = 0.2\ d_{11}^2$	$x_{01} = x_{11} - d_{11}$	-
21	d_{21}	$r_{21} = 375 - 10d_{21}$	$x_{11} = x_{21} + d_{21}$	-
2	d_2	$r_2 = 600 - 100x_2 + d_2$	$x_1 = x_2 + d_2$	-
3	d_3	$r_3 = 0.1\ (50 - d_3)^2$	$x_2 = x_3 + d_3$	-

Substituting the constraint into the recursion equation,we have

$$f_{11}(x_{11}, x_{01}) = 0.2\ (x_1 - x_{01})^2$$

At stage 21, we see that

$$f_{21}(x_{21}, x_{01}) = \min_{d_{21}} \{375 - 10d_{21} + 0.2\ (x_{21} + d_{21} - x_{01}{}^2)\}$$

Differentiating the term in brackets with respect to d_{21} and setting the derivative to zero yields

$$d_{21}(x_{21}, x_{01}) = 25 + x_{01} - x_{21}$$

and $$f_{21}(x_{21}, x_{01}) = 250 - 10x_{01} + 10x_{21}$$

Now, at stage 1 the optimal branch return is absorbed. Thus,

$$f_{1+21}(x_1, x_{21}) = \min_{x_{01},\ d_1} [0.1\ d_1{}^2 + 250 - 10x_{01} + 10x_{21}]$$

$$\text{s.t.} \quad d_1 = x_1 + x_{01}$$

Substituting the constraint into the foregoing recursion equation, we obtain

$$f_{1+21}(x_1, x_{21}) = \min_{x_{01}} [0.1(x_1 + x_{01})^2 + 250 - 10x_{01} + 10x_{21}]$$

The optimal value of x_{01} is readily found to be

$$x_{01}(x_1, x_{21}) = 50 - x_1$$

with $$d_1(x_1, x_{21}) = 50$$

and $$f_{1+21}(x_1, x_{21}) = 10(x_1 + x_{21})$$

At stage 2, we have

$$f_{2+21}(x_2, x_{21}) = \min_{d_2} [600 - 100x_2 + d_2 + 10(x_2 + d_2 + x_{21})]$$

The optimal value of d_2 in the above recursion equation is independent of the stage inputs x_2 and x_{21}. So,if we assume the feasible region of d_2 as

$$d_2 \in [\underline{d}_2, \bar{d}_2]$$

then, we have the optimal value of d_2 as

$$d_2 = \underline{d}_2$$

and $$f_{2+21}(x_2, x_{21}) = 600 + 11\underline{d}_2 = 90x_2 + 10x_{21}$$

Finally at stage 3, the recursion equation becomes

$$f_{3+21}(x_3, x_{21}) = \min_{d_3} [0.1 \ (50 - d_3)^2 + 600 + 11\underline{d}_2 - 90 \ (x_3 + d_3) + 10x_{21}]$$

It is easy to see that $d_3 = 500$ is the solution with

$$f_{3+21}(x_3, x_{21}) = 24150 + \underline{d}_2 - 90x_3 + 10x_{21}.$$

Retracing our analysis, we find the remainder of the optimal solution which is summarized as follows:

Table 3.6: Optimal Solution to the System of Fig. 3.5 and Table 3.5

Stage	Optimal Input	Optimal Decision	Optimal Return
1	$x_1 = x_3 + 500 + \underline{d}_2$	$d_1 = 50$	250
	$x_{01} = x_3 - 300 - \underline{d}_2$	-	-
11	$x_{11} = x_3 - 425 = \underline{d}_2$	$d_{11} = 25$	125
21	x_{21}	$d_{21} = x_3 - x_{21} - 425 - \underline{d}_2$	$10x_3 + 10x_{21} + 4625 + 10\underline{d}_2$
2	$x_2 = x_3 + 500$	$d_2 = \underline{d}_2$	$-100x_3 = 49400 - \underline{d}_2$
3	x_3	$d_3 = 500$	20250

Let us next solve the same problem described in Table 3.5 with the following constraints added to the input and decision variables at each stage

$$\min \sum_{n=1}^{3} r_n + \sum_{m=1}^{2} r_{m1}$$

$$\text{s.t.} \quad 491 \le x_1 \le 510$$
$$491 \le x_2 \le 510$$

$$\begin{aligned}
0 &\le x_3 \le 5 \\
-464 &\le x_{01} \le 445 \\
-439 &\le x_{11} \le -420 \\
0 &\le x_{21} \le 5 \\
41 &\le d_1 \le 60 \\
0 &\le d_2 \le 5 \\
11 &\le d_3 \le 30 \\
11 &\le d_{11} \le 30 \\
-439 &\le d_{21} \le -420 \\
x_n, x_{ml}, d_n, d_{ml} &\in T, \; \begin{matrix} n = 1, 2, 3 \\ m = 1, 2 \end{matrix}
\end{aligned}$$

where T is the set of constraints in Table 3.5. Notice that this is akin to the problem solved earlier in that the objective function is similar. However, the stage return functions are different. The solution is effected using a modification of the algorithm developed earlier. The computational result is shown with the computer output in Table 3.7. Note also that from Table 3.5, the total optimal return of the system in Fig. 3.5 becomes

$$10x_{21} - 90x_3 + 11\underline{d}_2 = 23150.$$

With the constraints given in the above problem, the optimal solution is obtained when

$$x_{21} = 0, \quad x_3 = 5, \text{ and } \underline{d}_2 = 0.$$

Further, note that in the computer output the problem was solved by changing the minimization of the objective function to the maximization of the total return.

The computational efficiency of our converging branch algorithm is measured with the problem by changing the number of discretizations of the variables. The problem above was solved with six discretizations in the variables x_{21}, x_3 and d_2 and 20 discretizations for all other variables.

Table 3.7: Computer Output of Constrained Example Problem

```
"RETURN TABLE AT STAGE 1"

X11 I   1    2    3    4    5    6    7    8    9   10
--------------------------------------------------------
  1 I   0.   1.   4.   9.   0.   0.   0.   0.   0.   0.
  2 I   0.   0.   1.   4.   9.   0.   0.   0.   0.   0.
  3 I   0.   0.   0.   1.   4.   9.   0.   0.   0.   0.
  4 I   0.   0.   0.   0.   1.   4.   9.   0.   0.   0.
  5 I   0.   0.   0.   0.   0.   1.   4.   9.   0.   0.
  6 I   0.   0.   0.   0.   0.   0.   1.   4.   9.   0.
  7 I   0.   0.   0.   0.   0.   0.   0.   1.   4.   9.
  8 I   0.   0.   0.   0.   0.   0.   0.   0.   1.   4.
  9 I   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.
 10 I   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
--------------------------------------------------------

"DECISION TABLE AT STAGE 1"

X11 I   1    2    3    4    5    6    7    8    9   10
--------------------------------------------------------
  1 I   0.   1.   2.   3.   0.   0.   0.   0.   0.   0.
  2 I   0.   0.   1.   2.   3.   0.   0.   0.   0.   0.
  3 I   0.   0.   0.   1.   2.   3.   0.   0.   0.   0.
  4 I   0.   0.   0.   0.   1.   2.   3.   0.   0.   0.
  5 I   0.   0.   0.   0.   0.   1.   2.   3.   0.   0.
  6 I   0.   0.   0.   0.   0.   0.   1.   2.   3.   0.
  7 I   0.   0.   0.   0.   0.   0.   0.   1.   2.   3.
  8 I   0.   0.   0.   0.   0.   0.   0.   0.   1.   2.
  9 I   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.
 10 I   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
--------------------------------------------------------

"RETURN TABLE AT STAGE 2"

X12 I   1    2    3    4    5    6    7    8    9   10
--------------------------------------------------------
  1 I   9.   9.   9.   9.  10.  13.  18.   9.   9.   9.
  2 I   9.   9.   9.   9.   9.  10.  13.  18.   9.   9.
  3 I   9.   9.   9.   9.   9.   9.  10.  13.  18.   9.
  4 I   9.   9.   9.   9.   9.   9.   9.  10.  13.  18.
  5 I   9.   9.   9.   9.   9.   9.   9.   9.  10.  13.
  6 I   9.   9.   9.   9.   9.   9.   9.   9.   9.  10.
  7 I   9.   9.   9.   9.   9.   9.   9.   9.   9.   9.
  8 I   9.   9.   9.   9.   9.   9.   9.        9.   9.
  9 I        9.   9.   9.   9.   9.   9.   9.   9.   9.
 10 I        9.   9.   9.   9.   9.   9.   9.   9.   9.
--------------------------------------------------------
```

Table 3.8 shows the storage requirement and the computational time for different combinations of discretizations. We see that the computational storage requirement is clearly affected by the number of discretizations of the branch output variable x_{01} as well as the other variables.

Table 3.8 Computational Storage Requirement and CPU Time for the Converging Branch System

Number of discretizations of all others	Number of discretizations of x_{01}			
	5	10	15	20
20	5,712 (.709)	10,322 (.989)	14,932 (1,313)	19,542 (1.674)
30	5,852 (1.077)	10,562 (1.834)	14,292 (2.484)	19,982 (3.219)
40	5,992 (1,729)	10,802 (2.913)	15,612 (4.077)	20,422 (5.201)

* Numbers in the ( ) represent the CPU times (seconds)
* The storage requirement is measured with the number of elements in the arrays. actually used.

3.6 Computational Efficiency of the Converging Branch Algorithm

The computer program of the converging branch algorithm is modified so that any integer values are permitted for input and decision variables at each stage in the system.

Assuming that the input (x_i, x_{i1}) and decision variables (d_i, d_{i1}) at each stage have their lower and upper bounds such that

$$\ell_i \leq x_i \leq u_i \qquad i = 1, 2, \ldots, N$$

$$\ell d_i \leq d_i \leq ud_i \qquad i = 1, 2, \ldots, N$$

$$\ell i_1 \leq x_{i1} \leq u_{i1} \qquad i = 1, 2, \ldots, M$$

$$\ell d_{i1} \leq d_{i1} \leq ud_{i1} \qquad i = 1, 2, \ldots, M$$

$$\ell_{01} \leq x_{01} \leq u_{01}$$

then the computational storage requirement can be analyzed as follows:

The main storage requirement in the converging branch algorithm is for the decision tables and the return tables. Since the decision table for the main serial process is constructed at each stage, the computational storage depends only on the number of discretizations of each decision variables. Hence, the decision table for the main serial process has the following storage requirement:

(Number of stages in the main serial process) multiplyed by (maximum number of discretizations of the input variables in the main serial process) $= N \times \max_{1 \leq i \leq N} (u_i - \ell_i + 1)$

The amount needed for the branch system is a little complex. Since at each stage of the branch, the transformation function has to satisfy the constraint given by the branch output (i.e, $\ell_{01} \leq x_{01} \leq u_{01}$) the storage requirement for the decision table becomes

Number of stages in the branch) multiplyed by Maximum number of discretizations of the input in the branch) multiplied by Number of discretizations of x_{01}) $= M \times \max_{1 \leq i \leq M} (u_{i1} - \ell_{i1} + 1) \times (u_{01} - \ell_{01} + 1)$

Another major source of storage requirement is due to the return table. Since the system is solved using a backward recursion equation, we need to revise the return at each stage with respect to the return at the previous stage. Hence, the storage requirement for the return table at the main serial process becomes

$$2 \times \{[\max_{1 \leq i \leq N} u_i] - [\min_{1 \leq i \leq N} \ell_i + 1]\}$$

Similarly, the computational storage of the return table for the branch system is given by

$$2 \times \{[\max_{i} u_{i1}] - [\min_{i} \ell_{i1} + 1]\} \times (u_{01} - \ell_{01} + 1)$$

Notice that the storage requirement of the return tables are largely dependent on the size of the input variables each of which is a function of the decision variable and input at the previous stage.

To illustrate the computational requirement of the converging branch system, we solved a set of problems with different numbers of stages both in the main serial process and in the branch. Table 3.9 shows the storage requirement and computational time for each case. In each case, a set of simple transformations and return functions at each stage was generated. Five discretizations for the branch output variable x_{01} and ten for other variables were used. The lowerbounds of all decision variables were assumed to be zero, and a reasonable lowerbound for each input variable was given.

We can see that the storage anu the CPU time increase are very sensitive to the number of stages in the branch system.

Table 3.9: Storage Requirement and CPU Times for Different Number of Stages

N = Number of stages in the serial process	M = Number of Stages in the Branch		
	2	3	4
5	470 (.411)	620 (.442)	770 (.566)
10	620 (.448)	770 (.499)	920 (.582)
15	770 (.474)	920 (.525)	1070 (.605)

* Numbers in ( ) represent the CPU times in seconds.
* The storage requirement is measured with the number of elements in the arrays.

10	[illegible]	[illegible]	5.	[illegible]	5.	5.	5.	5.	5.	5.	5.

"OPTIMAL BRANCH INPUT & RETURN FOR EACH X01"

X01	INPUT	RETURN"
1	10.	9.
2	10.	9.
3	10.	9.
4	10.	9.
5	1.	10.
6	1.	13.
7	1.	18.
8	2.	16.
9	3.	18.
10	4.	18.

"RETURN TABLE FOR MAIN PROCESS FOR EACH VALUE OF X0"

STAGE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.
2	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	18.	0.	0.	9.
3	45.	45.	45.	45.	45.	45.	45.	45.	45.	45.	45.	45.	45.	45.	45.	45.	45.	40.	37.	36.	36.	36.	36.	31.	28.	27.	27.	27.	[illegible]	27.
4	54.	54.	54.	54.	54.	54.	54.	54.	54.	54.	54.	54.	54.	54.	49.	36.	35.	[illegible]	45.	45.	40.	37.	36.	36.	36.	36.	36.	31.	[illegible]	9.
5	63.	63.	63.	63.	63.																									

```
*DECISION TABLE FOR MAIN PROCESS FOR EACH VALUE OF  U

STAGE I   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  [illegible]
---------------------------------------------------------------------------------------------------
    1 I  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  [illegible]
    2 I  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  [illegible]
    3 I  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  [illegible]
    4 I  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
    5 I  3.  3.  3.  3.  3.
---------------------------------------------------------------------------------------------------

*OPTIMAL INPUT VOL FROM BRANCH  TO JUNCTION*

  YS I   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  [illegible]
---------------------------------------------------------------------------------------------------
 VOL I 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. [illegible]
---------------------------------------------------------------------------------------------------

*STAGE  INPUT  DEFTS  RETURN*

   [illegible]   [illegible]   [illegible]   [illegible]
   [illegible]   8   3.0   9.0
   [illegible]   [illegible]   3.0   [illegible]
   [illegible]   [illegible]   3.0   [illegible]
   1   [illegible]   3.0   [illegible]
   [illegible]   [illegible]   3.0   [illegible]
   [illegible]   [illegible]   3.0   [illegible]
```

Chapter 4

ANALYSIS OF FEEDFORWARD LOOP SYSTEMS

4.1 The Basic Structure

A feedforward loop system is akin to the diverging branch system in which the branch output feeds into the main serial subsystem at state j. Thus, it may be viewed as a simple combination of diverging and converging systems. The divergence occurs at stage k while the convergence takes place at node j, j < k. The basic structure is diagrammed in Fig. 4.1.

Let the transformations and returns which are the same as in the usual serial ones for all stages other than j and k be defined as follows:

$$x_{n-1} = t_n(x_n, d_n)$$

$$r_n = r_n(x_n, d_n) \qquad , \quad n = 1, \ldots, N, \; n \neq j$$

$$x_{j-1} = t_j(x_{01}, x_j, d_j)$$

$$r_j = r_j(x_{01}, x_j, d_j)$$

$$x_{m-1,1} = t_{m1}(x_{m1}, d_{m1})$$

$$r_{m1} = r_{m1}(x_{m1}, d_{m1}) \qquad , \quad m = 1, \ldots, M$$

$$x_{M1} = t_{k1}(x_k, d_k)$$

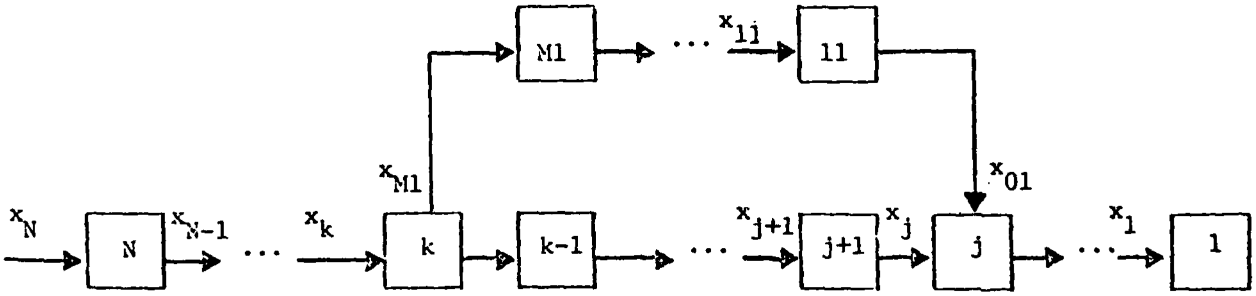


Figure 4.1: A Feedforward Loop System

Clearly, this is a more difficult system to treat than any of the systems discussed so far. A fundamental observation which supports this viewpoint is the fact that the branch is both diverging and converging. Thus, in nature the branch input x_{M1} as well as the output x_{01} affects the return from the serial system. As a consequence of this important fact, if the branch is optimized separately as a serial system, its optimal return must be determined as a function of both its input and output. In effect, we have a two point boundary value problem. There are thus at least two possible routes to the computational scheme. The optimal branch return can be absorbed into the main serial system either at the converging stage j or the diverging stage k. In either case, it must be noted that a two state variable dynamic programming problem results for the branch optimization. To minimize the computational burden, adroit schema must be sought to reduce the vector optimization problems to that of a series of one variable optimization problems.

4.2 The Optimization Procedure

Here, we will give a procedure which solves the feedforward loop system in which the absorption of the branch occurs at stage k. Nemhauser [22] describes both approaches but indicates why the absorption at the diverging stage is the preferred procedure. In this procedure, there are essentially two main steps involving the loop system and the main serial system.

4.2.1) Optimization of the Loop System (from stage 11 to M1)

The branch consisting of stages 11 through M1 is optimized to find $f_{M1}(x_{M1}, x_{01})$. This procedure is the same as the one in the converging branch system treated earlier. Notice that this is a two state variable dynamic program.

4.2.2) Optimization of the Main Serial System

The following five phases in the procedure may be considered.

4.2.2-i) For the stages from 1 to j - 1 (from node 1 to node preceding the convergent node j)

The optimal return from stage 1 through j - 1 is obtained by using the usual recursive procedure, i.e.,

$$f_1(x_1) = \max_{d_1} r_1(x_1, d_1)$$

$$f_n(x_n) = \max_{d_n} [r_n(x_n, d_n) + f_{n-1}(t_n(x_n, d_n))], \; n = 2, \ldots, j-1$$

4.2.2-ii) For stages from j to k - 1 (from convergent node to node proceeding divergent node k)

Since the absorption of the loop is assumed to occur at stage k, the optimization of x_{01} is deferred to stage k. Hence, x_{01} is carried as a state variable in stages j through k - 1. Let the optimal return be defined as $f_j(x_j, x_{01})$.

The resultant recursive equations are

$$f_j(x_j, x_{01}) = \max_{d_j} [r_j(x_j, x_{01}, d_j) + f_{j-1}(t_j(x_j, x_{01}, d_j))]$$

and
$$f_n(x_n, x_{01}) = \max_{d_n} [r_n(x_n, d_n) + f_{n-1}(t_n(x_n, d_n), x_{01})]$$
$$n = j+1, \ldots, k-1$$

4.2.2-iii) Optimization of x_{01} at stage k (absorption of divergent node)

At stage k, the optimal loop return $f_{M1}(x_{M1}, x_{01})$ is absorbed into the main serial process and we have the following recursion equation:

$$f_{k+M1}(x_k) = \max_{x_{01}, d_k} [r_k(x_k, d_k) + f_{k-1}(t_k(x_k, d_k), x_{01}) + f_{M1}(t_{k1}(x_k, d_k), x_{01})]$$

At this stage, the optimal branch output $x_{01}(x_k)$ is obtained as a function of the input variable x_k, of the stage k.

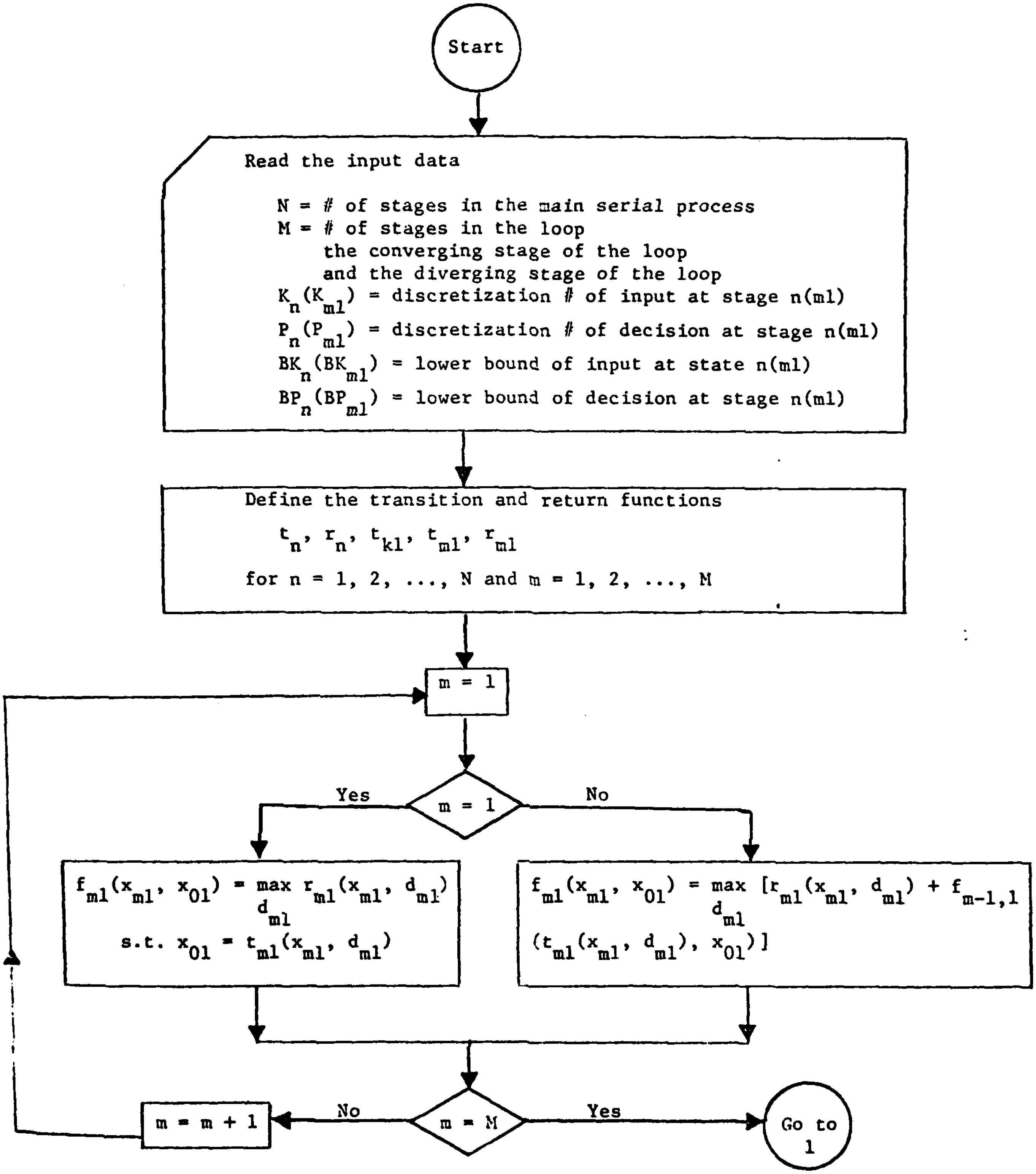
4.2.2-iv) <u>For the stages from k + 1 through N</u> (remainder of the serial system)

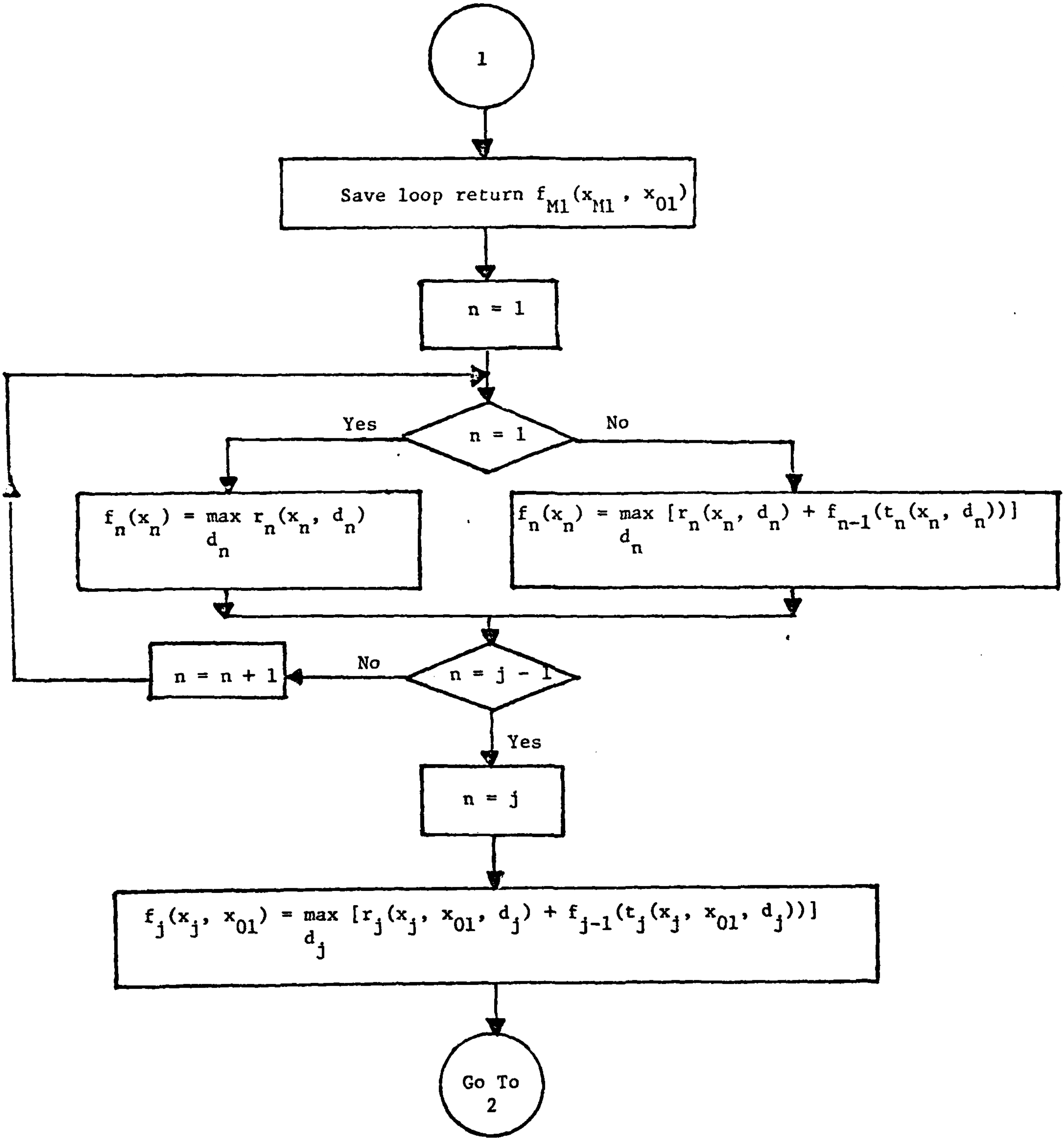
The recursion equation for stages k + 1 through N is given as usual by

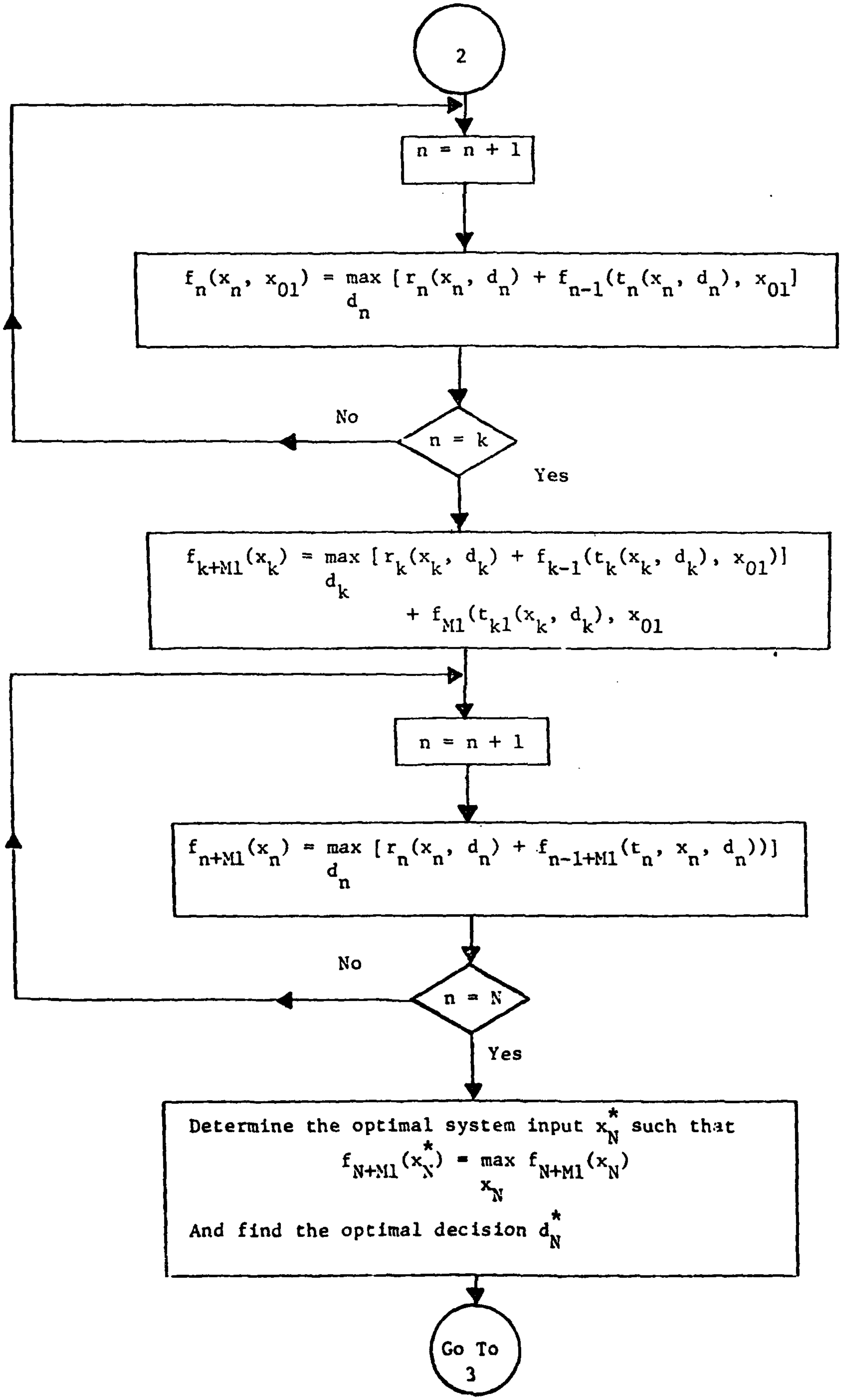
$$f_{n+M1}(x_n) = \max_{d_n} [r_n(x_n, d_n) + f_{n-1+M1}(t_n(x_n, d_n))],$$
$$n = k+1, \ldots, N$$

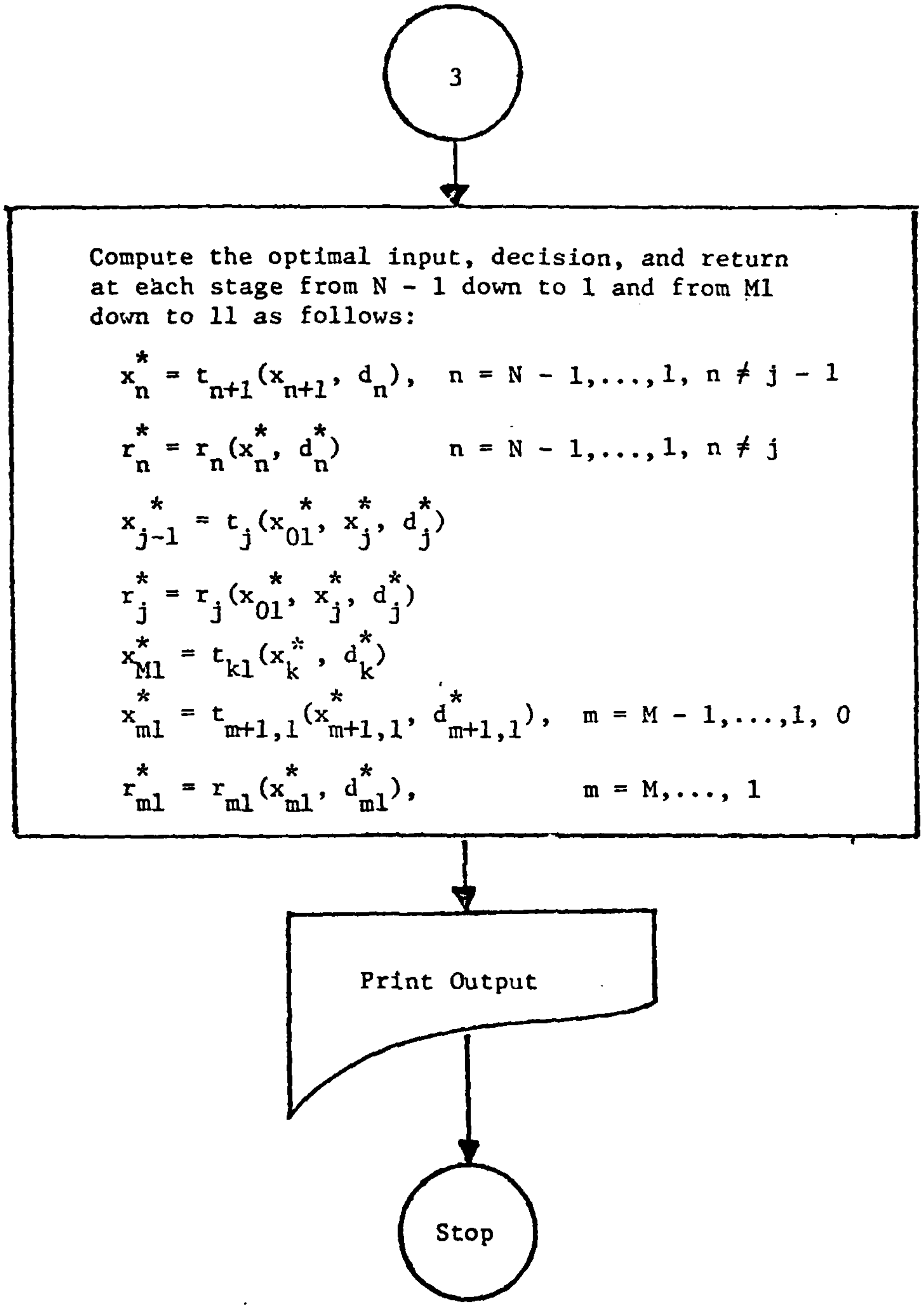
This concludes the optimization phases for the situation in which absorption takes place at the divergent node. When it takes place at the convergent node j, a different set of recursion equations is needed. In general, the same number of computations is required in either case. The main difference in the procedure results from the consideration of where x_{01} is optimized. In the convergent node absorption variety, the optimization is done earlier in the analysis with x_{M1} being carried as a state variable from stages j + 1 to k - 1 whereas x_{01} is carried as a state variable during similar periods when the optimization is done at the divergent node k as just described.

Figure 4.2: Flow Chart for the Feedforward Loop System


Start
Read the input data
N = # of stages in the main serial process
M = # of stages in the loop
the converging stage of the loop
and the diverging stage of the loop
$K_n(K_{m1})$ = discretization # of input at stage n(m1)
$P_n(P_{m1})$ = discretization # of decision at stage n(m1)
$BK_n(BK_{m1})$ = lower bound of input at state n(m1)
$BP_n(BP_{m1})$ = lower bound of decision at stage n(m1)
Define the transition and return functions
$t_n, r_n, t_{k1}, t_{m1}, r_{m1}$
for n = 1, 2, ..., N and m = 1, 2, ..., M
m = 1
m = 1
Yes
No
$f_{m1}(x_{m1}, x_{01}) = \max_{d_{m1}} r_{m1}(x_{m1}, d_{m1})$
s.t. $x_{01} = t_{m1}(x_{m1}, d_{m1})$
$f_{m1}(x_{m1}, x_{01}) = \max_{d_{m1}} [r_{m1}(x_{m1}, d_{m1}) + f_{m-1,1}$
$(t_{m1}(x_{m1}, d_{m1}), x_{01})]$
m = M
No
Yes
m = m + 1
Go to 1


1
Save loop return $f_{M1}(x_{M1}, x_{01})$
$n = 1$
$n = 1$
Yes
No
$f_n(x_n) = \max_{d_n} r_n(x_n, d_n)$
$f_n(x_n) = \max_{d_n} [r_n(x_n, d_n) + f_{n-1}(t_n(x_n, d_n))]$
$n = n + 1$
No
$n = j - 1$
Yes
$n = j$
$f_j(x_j, x_{01}) = \max_{d_j} [r_j(x_j, x_{01}, d_j) + f_{j-1}(t_j(x_j, x_{01}, d_j))]$
Go To 2


2
n = n + 1
$f_n(x_n, x_{01}) = \max_{d_n} [r_n(x_n, d_n) + f_{n-1}(t_n(x_n, d_n), x_{01}]$
No
n = k
Yes
$f_{k+M1}(x_k) = \max_{d_k} [r_k(x_k, d_k) + f_{k-1}(t_k(x_k, d_k), x_{01})]$
$+ f_{M1}(t_{k1}(x_k, d_k), x_{01}$
n = n + 1
$f_{n+M1}(x_n) = \max_{d_n} [r_n(x_n, d_n) + f_{n-1+M1}(t_n, x_n, d_n))]$
No
n = N
Yes
Determine the optimal system input x_N^* such that
$f_{N+M1}(x_N^*) = \max_{x_N} f_{N+M1}(x_N)$
And find the optimal decision d_N^*
Go To 3



Compute the optimal input, decision, and return at each stage from N - 1 down to 1 and from M1 down to 11 as follows:

$$x_n^* = t_{n+1}(x_{n+1}, d_n), \quad n = N - 1,\dots,1,\ n \neq j - 1$$

$$r_n^* = r_n(x_n^*, d_n^*) \quad n = N - 1,\dots,1,\ n \neq j$$

$$x_{j-1}^* = t_j(x_{01}^*, x_j^*, d_j^*)$$

$$r_j^* = r_j(x_{01}^*, x_j^*, d_j^*)$$

$$x_{M1}^* = t_{k1}(x_k^*, d_k^*)$$

$$x_{m1}^* = t_{m+1,1}(x_{m+1,1}^*, d_{m+1,1}^*), \quad m = M - 1,\dots,1,\ 0$$

$$r_{m1}^* = r_{m1}(x_{m1}^*, d_{m1}^*), \quad m = M,\dots,\ 1$$

4.3 An Example of the Feedforward Loop System

The flow chart for the numerical solution of the feedforward loop system is shown in Figure 4.2. To illustrate the algorithm, a computer program was written and implemented using the test problem described in Figure 4.3. In the example, it is desired to maximize the sum of the stage returns for the problem with the structure given in Figure 4.3 using the data in Table 4.1. In this example, the convergent stage is node 2 and the divergent stage is node 5. The loop contains nodes 31, 21, and 11.

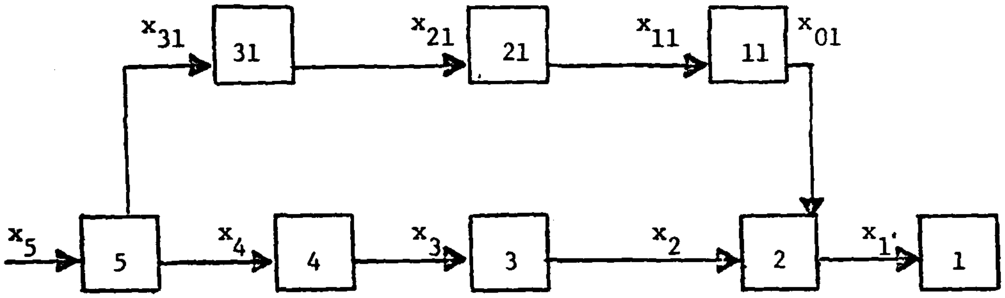


Figure 4.3: A Feedforward Loop System Example

Table 4.1: The Return and Transition Function for the System of Fig. 4.3

Stage	Decision	Return	Transition	Constraints
1	d_1	$r_1 = x_1 + d_1$	-	
2	d_2	$r_2 = x_{01} + x_2 + d_2$	$(x_{01} + x_2)/2 + d_2$	
11	d_{11}	$r_{11} = x_{11} + d_{11}^2$	$x_{11} + d_{11}$	$0 \le x_i \le 15,\ i=1,\dots,4$
21	d_{21}	$r_{21} = x_{21} + d_{21}^2$	$x_{21} + d_{21}$	$0 \le x_{i1} \le 10,\ i=0, 1, 2, 3$
31	d_{31}	$r_{31} = x_{31} + d_{31}^2$	$x_{31} + d_{31}$	$0 \le x_5 \le 3$
3	d_3	$r_3 = x_3 + d_3$	$x_3 + d_3$	$0 \le d_i \le 2,\ i=1,\dots,5$
4	d_4	$r_4 = x_4 + d_4$	$x_4 + d_4$	$0 \le d_{i1} \le 2,\ i=1, 2, 3$
5	d_5	$r_5 = x_5 + d_5$	$x_5 + d_5$	

Our problem can be formulated using the following mathematical form:

$$\max \sum_{n=1}^{5} r_n + \sum_{m=1}^{2} r_{m1}$$

$$\begin{aligned}
\text{s.t.} \quad & 0 \le x_n \le 15 && n = 1, \ldots, 4 \\
& 0 \le x_5 \le 3 && \\
& 0 \le x_{m1} \le 10 && m = 0, 1, \ldots, 3 \\
& 0 \le d_n \le 2 && n = 1, \ldots, 5 \\
& 0 \le d_{m1} \le 2 && m = 1, 2, 3
\end{aligned}$$

The computational results are shown in the computer output of Table 4.2 The optimal decision and return at each stage are summarized in Table 4.3.

Table 4.2: Computer Output of Feedforward Loop Example in Figure 4.3

```
 DECISION TABLE AT STAGE 11

0.    1.    2.************************************************************
**    0.    1.    2.*******************************************************
*******    0.    1.    2.**************************************************
************    0.    1.    2.*********************************************
*****************    0.    1.    2.****************************************
**********************    0.    1.    2.***********************************
***************************    0.    1.    2.******************************
********************************    0.    1.    2.*************************
*************************************    0.    1.    2.********************
******************************************    0.    1.    2.***************
***********************************************    0.    1.    2.**********


 DECISION TABLE AT STAGE 21

0.    1.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.
2.    0.    1.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.
2.    2.    0.    1.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.
2.    2.    2.    0.    1.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.
2.    2.    2.    2.    0.    1.    2.    2.    2.    2.    2.    2.    2.    2.    2.
2.    2.    2.    2.    2.    0.    1.    2.    2.    2.    2.    2.    2.    2.    2.
2.    2.    2.    2.    2.    2.    0.    1.    2.    2.    2.    2.    2.    2.    2.
2.    2.    2.    2.    2.    2.    2.    0.    1.    2.    2.    2.    2.    2.    2.
2.    2.    2.    2.    2.    2.    2.    2.    0.    1.    2.    2.    2.    2.    2.
2.    2.    2.    2.    2.    2.    2.    2.    2.    0.    1.    0.    1.    2.    2.
2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    0.    0.    0.    2.    2.


 DECISION TABLE AT STAGE 31

0.    1.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.
2.    0.    1.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.
2.    2.    0.    1.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.
2.    2.    2.    0.    1.    2.    2.    2.    2.    2.    2.    2.    2.    2.    2.
2.    2.    2.    2.    0.    1.    2.    2.    2.    2.    2.    2.    2.    2.    2.
2.    2.    2.    2.    2.    0.    1.    2.    2.    2.    2.    2.    2.    2.    2.
2.    2.    2.    2.    2.    2.    0.    1.    2.    2.    2.    2.    2.    2.    2.
2.    2.    2.    2.    2.    2.    2.    0.    1.    2.    2.    2.    2.    2.    2.
2.    2.    2.    2.    2.    2.    2.    2.    0.    1.    2.    2.    2.    2.    2.
2.    2.    2.    2.    2.    2.    2.    2.    2.    0.    1.    1.    1.    2.    2.
2.    2.    2.    2.    2.    2.    2.    2.    2.    2.    0.    0.    0.    2.    2.
```

Table 4.2 Continued

```
'DECISION TABLE FOR MAIN SERIAL PROCESS'

 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.

 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.

 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.

 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
 2.   2.   2.   2.   2.   2.   2.   2.   2.   1.   0.
```

Table 4.2 Continued

```
   2.    2.    2.    2.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
   0.    0.    0.    0.
 'OPTIMAL LOOP OUTPUT FOR EACH INPUT VALUE'
 'OF THE DIVERGING STAGE K'
    0       8
    1       9
    2      10
    3      11

 'STAGE  INPUT  DECIS  RETURN'

    5      3       2        5.
    4      5       2        7.
    3      7       2        9.
    2      9       2       22.
    1     12       2       14.
   31      5       2        9.
   21      7       2       11.
   11      9       2       13.

TOTAL OPTIMAL RETURN IS    90.
 EOI, 0 FILES, 1 RECS, 679 WORDS.
/
```

Table 4.3 : Optimal Input, Decision, and Return of the System in Figure 4.3

Stage	Opt. Input	Opt. Decision	Opt. Return
5	3	2	5
4	5	2	7
3	7	2	9
31	5	2	9
21	7	2	11
11	9	2	13
2	9 (11)*	2	22
1	12	2	14

* ( ) represents the input from the feedforward loop

4.4 Analysis of the Feedforward Loop System

The storage requirement of the feedforward loop system can be characterized by considering the following two aspects:

1. The storage requirement for the loop system can be analyzed as the converging branch system.
2. The storage requirement for the main serial system is much higher than that for the diverging or converging branch system. This is because x_{01} is carried as a state variable for stages j through k - 1.

Now, let U_i, ℓ_i, U_{i1} and ℓ_{i1} be defined as follows:

$$\ell_i \le x_i \le U_i \qquad i = 1, 2, \ldots, N$$

$$\ell_{i1} \le x_{i1} \le U_{i1} \qquad i = 0, 1, \ldots, M$$

Also, let K_i and K_{i1} be respectively the number of discretization levels for variables x_i and x_{i1}, and K and K_1 be defined as follows:

$$K = \max_{1 \le i \le N} K_i$$

$$K_1 = \max_{0 \le i \le M} K_{i1}$$

Since the loop optimization must be treated as an initial-final value problem as in the converging branch system, the storage requirement for the loop system becomes

$$SRL = MK_1K_{01} + 2\, K_{01}\, (\max_i U_{i1} - \min_i \ell_{i1})$$

The first term MK_1K_{01} of the above equation is the storage requirement for the decision value for each pair of input and branch output values. The second term is the storage requirement for the stage returns for each possible pair of input variable x_{i1} and branch output x_{01}.

Now, the analysis of the main serial system of the loop structure is different from the one for the converging branch system. In the converging branch system, the branch return is combined at the converging stage S with the returns from main serial process and then the branch output x_{01} is optimized as a function of x_S. However, in the loop structure, x_{01} cannot be optimized at the converging stage j, since all the recursive returns from stage j to k - 1 are affected by the variable x_{01}. Hence, x_{01} needs to be carried as a state variable for stages j through k - 1. Thus, the storage requirement for the main serial system becomes

$$SRMS = NKK_{01} + 2\, K_{01}\, (\max_i U_i - \min_i \ell_i)$$

Thus, the total storage requirement for the loop structure becomes

$$\begin{aligned} TRSL = {} & MK_1K_{01} + 2\, K_{01} (\max_i U_i - \min_i \ell_{i1}) \\ & + NKK_{01} + 2\, K_{01}\, (\max_i U_i - \min_i \ell_i) \end{aligned}$$

If we assume $K_1 = K$, $\max U_{i1} = \max U_i$ and $\min \ell_{i1} = \min \ell_i$, then the above requirement becomes

$$TRSL = (M + N)\ KK_{01} + 2\ K_{01}\ (\max U_i = \min \ell_i)$$

4.5 Computational Experiments with the Loop System

In Table 4.4 we will illustrate the computational requirement of the feedforward loop system. We solved a set of problems with different numbers of stages in the loop. For each problem the following input data were used.

$N = 5$

$K = K_1 = 15$

$K_{01} = 10$

$\max U_i = 14$

$\min \ell_i = 0$

As shown in the table, both the storage requirement and the CPU time can be represented as a linear function of the number of stages in the loop.

Table 4.4: Computer Storage Requirements for the Feedforward Loop System

Number of Stages in the Loop	Storage Requirements	CPU Time (Seconds)
2	1350	.602
3	1500	.694
4	1650	.748
5	1800	.875

Chapter 5

ANALYSIS OF FEEDBACK LOOP SYSTEMS

5.1 The Basic Structure and Algorithm

A feedback loop system is identical to a feedforward system except that the relative positions of stages j and k are reversed. In the feedback loop system as shown in Figure 5.1, $k < j$, where k is the diverging stage and j the converging stage.

The transition and return functions for the feedback loop are identical to those of the feedforward loop with the numbering of the stages as shown in Figure 5.1.

The recursion equations for the feedback loops are a little different because of the different positions of stages j and k. The optimization procedures for the stages in the loop as well as those in the main serial process except stages j and k are the same as in the feedforward loop systems. Hence, we will not give the detailed optimization procedures for the feedback loop systems. Instead, we will summarize the -lgorithm in the flow chart given in Figure 5.2.

In the algorithm described in the flow chart, the optimal loop return $f_{M1}(x_{M1}, x_{01})$ is combined with the main serial system at the diverging stage k, and stages $k + 1$ through $j - 1$ are optimized to give $f_{j-1+M1}(x_{j-1}, x_{01})$. The branch output x_{01} is then optimized at the converging stage j.

The analysis of the feedback loop system is not given since the complexity of this system is identical to that of the feedforward loop structure presented in section 4.4.

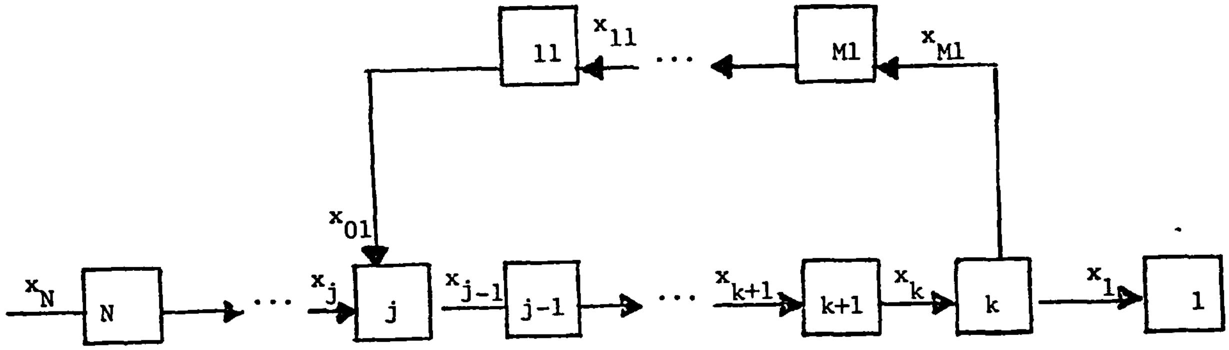
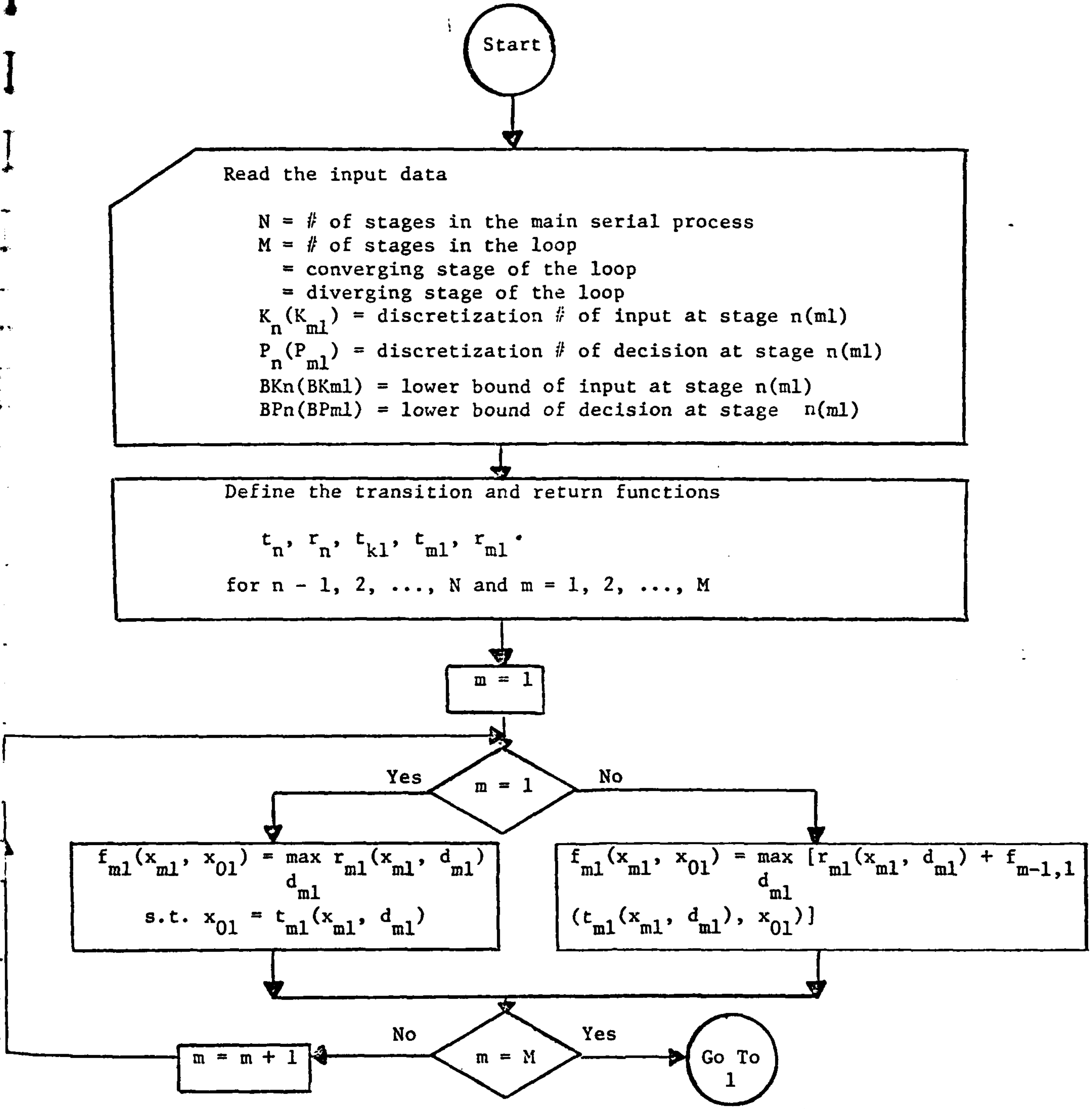
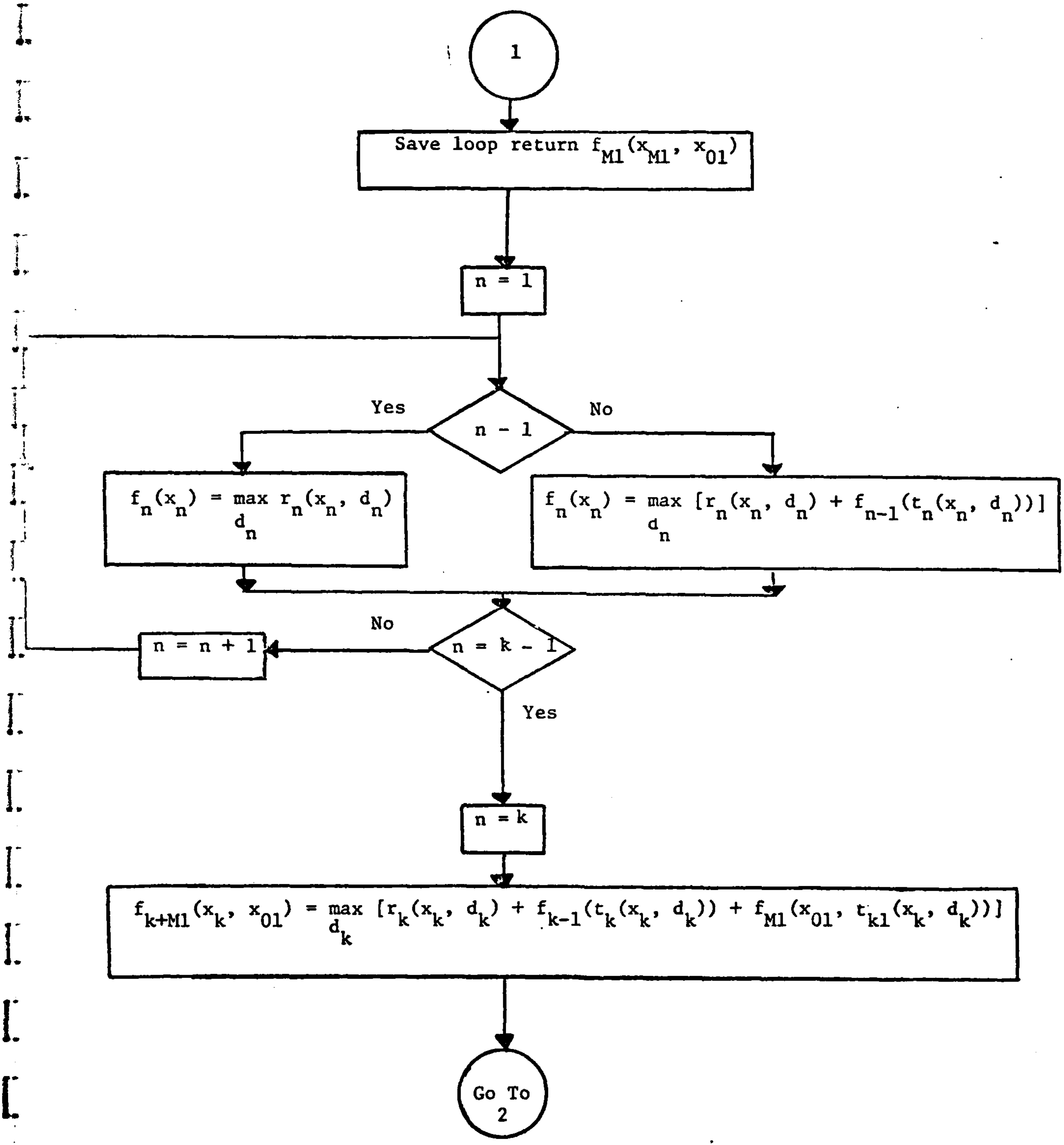
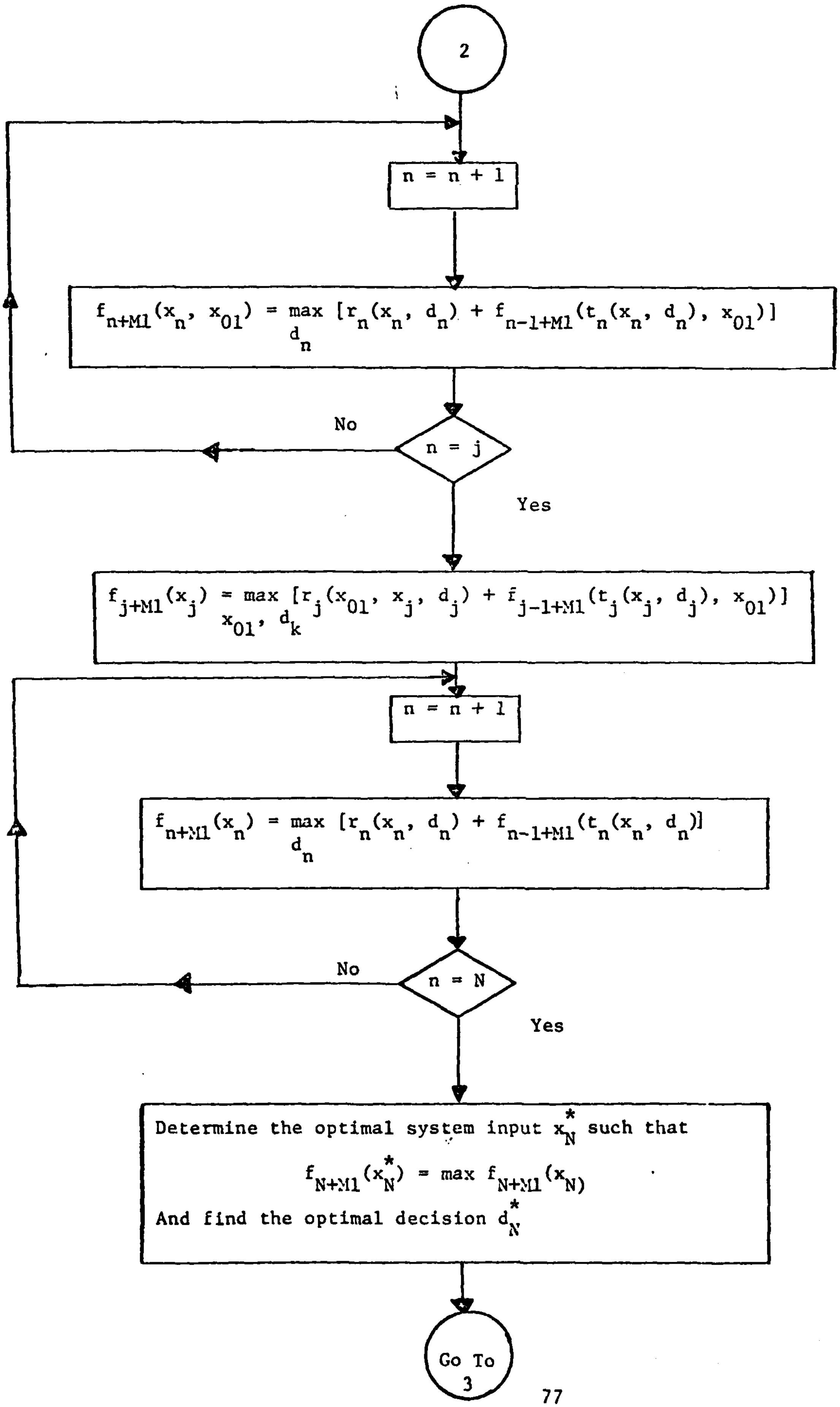


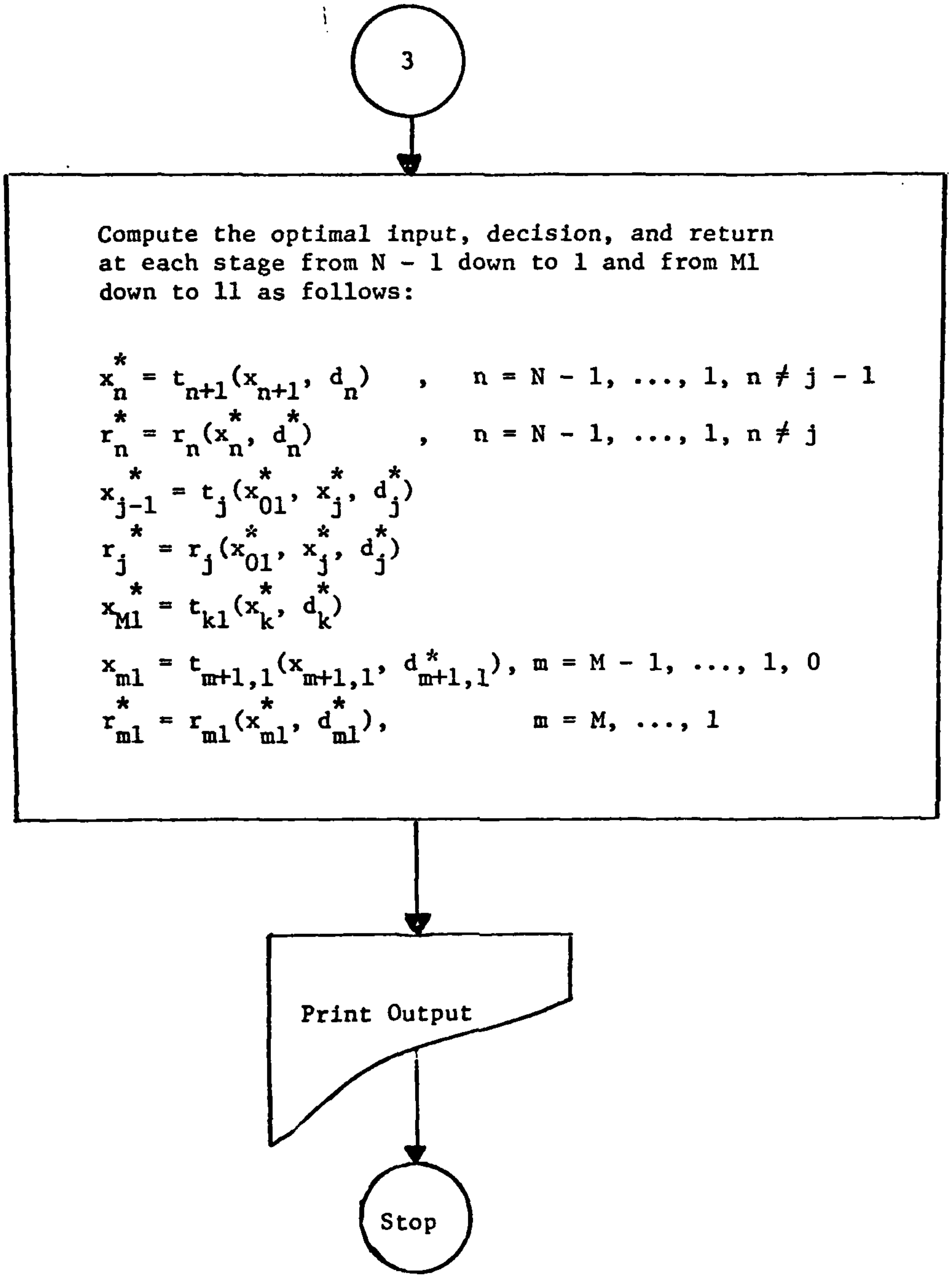
Figure 5.1: A Feedback Loop System

Figure 5.2: Flow Chart for the Feedback Loop System


Start
Read the input data
N = # of stages in the main serial process
M = # of stages in the loop
= converging stage of the loop
= diverging stage of the loop
$K_n(K_{ml})$ = discretization # of input at stage n(ml)
$P_n(P_{ml})$ = discretization # of decision at stage n(ml)
BKn(BKml) = lower bound of input at stage n(ml)
BPn(BPml) = lower bound of decision at stage n(ml)
Define the transition and return functions
t_n, r_n, t_{k1}, t_{ml}, r_{ml} .
for n - 1, 2, ..., N and m = 1, 2, ..., M
m = 1
m = 1
Yes
No
$f_{ml}(x_{ml}, x_{01}) = \max_{d_{ml}} r_{ml}(x_{ml}, d_{ml})$
s.t. $x_{01} = t_{ml}(x_{ml}, d_{ml})$
$f_{ml}(x_{ml}, x_{01}) = \max_{d_{ml}} [r_{ml}(x_{ml}, d_{ml}) + f_{m-1,1}$
$(t_{ml}(x_{ml}, d_{ml}), x_{01})]$
No
Yes
m = M
m = m + 1
Go To
1


1
Save loop return $f_{M1}(x_{M1}, x_{01})$
$n = 1$
Yes
$n - 1$
No
$f_n(x_n) = \max_{d_n} r_n(x_n, d_n)$
$f_n(x_n) = \max_{d_n} [r_n(x_n, d_n) + f_{n-1}(t_n(x_n, d_n))]$
No
$n = n + 1$
$n = k - 1$
Yes
$n = k$
$f_{k+M1}(x_k, x_{01}) = \max_{d_k} [r_k(x_k, d_k) + f_{k-1}(t_k(x_k, d_k)) + f_{M1}(x_{01}, t_{k1}(x_k, d_k))]$
Go To 2


2
$n = n + 1$
$f_{n+M1}(x_n, x_{01}) = \max_{d_n} [r_n(x_n, d_n) + f_{n-1+M1}(t_n(x_n, d_n), x_{01})]$
No
$n = j$
Yes
$f_{j+M1}(x_j) = \max_{x_{01}, d_k} [r_j(x_{01}, x_j, d_j) + f_{j-1+M1}(t_j(x_j, d_j), x_{01})]$
$n = n + 1$
$f_{n+M1}(x_n) = \max_{d_n} [r_n(x_n, d_n) + f_{n-1+M1}(t_n(x_n, d_n)]$
No
$n = N$
Yes
Determine the optimal system input x_N^* such that
$f_{N+M1}(x_N^*) = \max f_{N+M1}(x_N)$
And find the optimal decision d_N^*
Go To 3



Compute the optimal input, decision, and return at each stage from N - 1 down to 1 and from M1 down to 11 as follows:

$$x_n^* = t_{n+1}(x_{n+1}, d_n) \quad , \quad n = N - 1, \ldots, 1, n \neq j - 1$$

$$r_n^* = r_n(x_n^*, d_n^*) \quad , \quad n = N - 1, \ldots, 1, n \neq j$$

$$x_{j-1}^* = t_j(x_{01}^*, x_j^*, d_j^*)$$

$$r_j^* = r_j(x_{01}^*, x_j^*, d_j^*)$$

$$x_{M1}^* = t_{k1}(x_k^*, d_k^*)$$

$$x_{m1} = t_{m+1,1}(x_{m+1,1}, d_{m+1,1}^*), \quad m = M - 1, \ldots, 1, 0$$

$$r_{m1}^* = r_{m1}(x_{m1}^*, d_{m1}^*), \quad m = M, \ldots, 1$$

5.2 An Example of a Feedback Loop System

To illustrate the optimization of a feedback loop system, we consider an example with the structure shown in Figure 5.3 and the data given in Table 5.1. The objective is to maximize the sum of the stage returns.

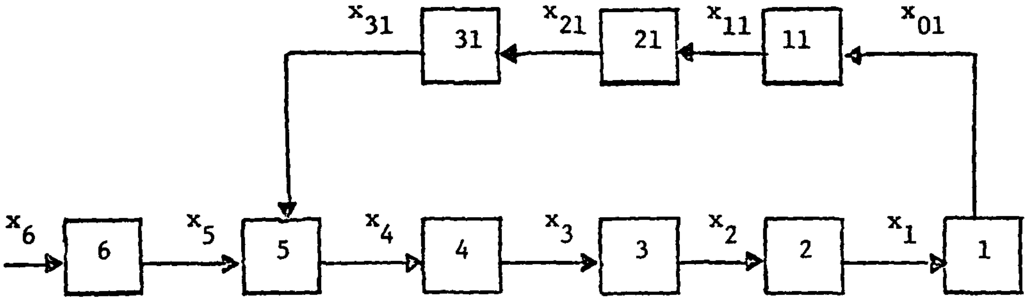


Figure 5.3: A Feedback Loop System

Table 5.1: The Return and Transition Functions of the System in Figure 5.3

Stage	Decision	Return	Transition	Constraints
11	d_{11}	$r_{11} = x_{11} + d_{11}^2$	$x_{01} + d_{11}$	
21	d_{21}	$r_{21} = x_{21} + d_{21}^2$	$x_{11} + d_{21}$	$0 \leq x_i \leq 25 \quad i \neq 6$
31	d_{31}	$r_{31} = x_{31} + d_{31}^2$	$x_{21} + d_{31}$	$0 \leq x_6 \leq 4$
1	d_1	$r_1 = x_1 + d_1$	$(x_1 + d_1^2) \quad 4$	$0 \leq x_{i1} \leq 15$
2	d_2	$r_2 = x_2 + d_2$	$x_2 + d_2$	$0 \leq x_{01} \leq 6$
3	d_3	$r_3 = x_3 + d_3$	$x_3 + d_3$	$0 \leq d_i \leq 3 \quad i = 2, \ldots, 6$
4	d_4	$r_4 = x_4 + d_4$	$x_4 + d_4$	$0 \leq d_i \leq 2$
5	d_5	$r_5 = x_5 + d_5$	$x_5 + d_5 + x_{31}$	$0 \leq d_{i1} \leq 2$
6	d_6	$r_6 = x_6 + d_6$	$x_6 + d_6$	

Table 5.2: Computer Output for Feedforward Loop Example in Figure 5.3

```
DECISION TABLE AT STAGE 11

0.   1.   2.************************************************************
**   0.   1.   2.*******************************************************
*******   0.   1.   2.**************************************************
************   0.   1.   2.*********************************************
*****************   0.   1.   2.****************************************
**********************   0.   1.   2.***********************************
***************************   0.   1.   2.******************************
********************************   0.   1.   2.*************************
*************************************   0.   1.   2.********************
******************************************   0.   1.   2.***************
***********************************************   0.   1.   2.**********
****************************************************   0.   1.   2.*****
*********************************************************   0.   1.   2.
**************************************************************   0.   1.
*******************************************************************   0.
```

```
DECISION TABLE AT STAGE 21

0.   1.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
2.   0.   1.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
2.   2.   0.   1.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
2.   2.   2.   0.   1.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
2.   2.   2.   2.   0.   1.   2.   2.   2.   2.   2.   2.   2.   2.   2.
2.   2.   2.   2.   2.   0.   1.   2.   2.   2.   2.   2.   2.   2.   2.
2.   2.   2.   2.   2.   2.   0.   1.   2.   2.   2.   2.   2.   2.   2.
2.   2.   2.   2.   2.   2.   2.   0.   1.   2.   2.   2.   2.   2.   2.
2.   2.   2.   2.   2.   2.   2.   2.   0.   1.   2.   2.   2.   2.   2.
2.   2.   2.   2.   2.   2.   2.   2.   2.   0.   1.   2.   2.   2.   2.
2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   0.   1.   2.   2.   2.
2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   0.   1.   2.   2.
2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   0.   1.   2.
2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   0.   1.
2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   0.
```

```
DECISION TABLE AT STAGE 31

0.   1.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
2.   0.   1.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
2.   2.   0.   1.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
2.   2.   2.   0.   1.   2.   2.   2.   2.   2.   2.   2.   2.   2.   2.
2.   2.   2.   2.   0.   1.   2.   2.   2.   2.   2.   2.   2.   2.   2.
2.   2.   2.   2.   2.   0.   1.   2.   2.   2.   2.   2.   2.   2.   2.
2.   2.   2.   2.   2.   2.   0.   1.   2.   2.   2.   2.   2.   2.   2.
```

Table 5.2 Continued

```
'DECISION TABLE FOR MAIN SERIAL PROCESS'

 1.  1.  1.  0.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.
 2.  2.  2.  2.  1.  1.  1.  0.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.
 1.  1.  1.  2.  2.  2.  2.  2.  1.  1.  1.  0.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.
 2.  2.  2.  2.  1.  1.  1.  2.  2.  2.  2.  2.  1.  1.  1.  0.  2.  2.  2.  2.  2.  2.  2.  2.  1.
 2.  2.  2.  2.  2.  2.  2.  2.  1.  1.  1.  2.  2.  2.  2.  2.  1.  1.  1.  0.  2.  2.  2.  2.  1.
 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.  1.  1.  2.  2.  2.  2.  2.  1.  1.  1.  0.  1.
 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.  1.  1.  2.  2.  2.  2.  2.  1.
 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.  1.  1.  2.  1.
 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.
 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.
 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.
 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.
 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.
 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.
 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  1.

 3.  2.  1.  0.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  2.  1.  0.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  2.  1.  0.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  2.  1.  0.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  2.  1.  0.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
 3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.  3.
```

Table 5.2 Continued

3.	2.	1.	0.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.
3.	3.	3.	3.	3.	2.	1.	0.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.
3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.	3.	3.	3.	3.	3.	3.	3.	3.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.	3.	3.	3.	3.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.

3.	2.	1.	0.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	2.	1.	0.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.
3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	3.	2.	1.	0.

Table 5.2 Continued

```
 3.   3.   3.   3.   3.   3.   3.   3.   3.   3.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   6.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.

 3.   3.   3.   3.   3.

'OPTIMAL LOOP OUTPUT FOR EACH INPUT VALUE'
'OF THE CONVERGING STAGE J'
   0      11
   1      11
   2      11
   3      11
   4      11
   5      11
   6      11
   7      12
   8      12
   9      12

'STAGE  INPUT  DECIS  RETURN'

   6      4        3        7.
   5      7        3       22.
   4     11        3       14.
   3     14        3       17.
   2     17        3       20.
   1     20        2       22.
  31      6        2       10.
  21      8        2       12.
  11     10        2       14.

TOTAL OPTIMAL RETURN IS    138.
 EOI. 0 FILES. 1 RECS. 1187 WORDS.
/
```

In Table 5.3 we display a summary of the results of this example problem while the decision table at each stage is shown in the computer output of Table 5.2.

Table 5.3: Optimal Input, Decision, and Return from Table 5.2

Stage	Opt. Input	Opt. Decision	Opt. Return
31	6	2	10
21	8	2	12
11	10	2	14
6	4	3	7
5	7 (12)*	3	22
4	11	3	14
3	14	3	17
2	17	3	20
1	20	2	22

* ( ) represents the optimal input from feedback loop to stage 5

Chapter 6

SOLUTION OF SPECIALLY STRUCTURED NONSERIAL NETWORKS VIA THE IMBEDDED STATE SPACE DYNAMIC PROGRAMMING

6.1 Introduction

In this chapter, we want to show how certain nonserial networks with special structures can be solved using some recently developed dynamic programming algorithms. The structure in question refers not so much to the network configuration as it does to the nature of the return functions and the constraint spaces. For example, when the return functions are discontinuous and, in particular, are (transformable to) step functions, the imbedded state space dynamic programming developed by Esogbue and Morin [20] is an especially potent procedure. This method mitigates the usual curse of dimensionality problem inherent in most dynamic programming solutions by reducing an M-dimensional search problem to a one dimensional dynamic programming over a sequence of imbedded state spaces.

An excellent application of this procedure to the multidimensional knapsack problem was discussed by Morin and Esogbue while Morin and Marsten [21] give the details of an algorithm developed for the computational solution of large scale dynamic programming problems.

6.2 Outline of the Imbedded State Space Approach (ISSA)

The key to this approach is the exploitation of the discontinuity preserving properties of the maximal convolution to transform a search over the entire state space to one restricted to a set of imbedded state spaces. This proposition is propounded in [20]. We outline the procedure in the sequel.

Consider the following multidimensional knapsack problem in which the $r_n(\cdot)$ are discontinuous and furthermore step functions:

$$\max \sum_{j=1}^{N} r_j(x_j) \qquad (1)$$

$$\text{s.t.} \sum_{j=1}^{N} g_{ij}(x_j) \leq b_i \quad , \quad i = 1, 2, \ldots, M \qquad (2)$$

$$x_j \in S_j \quad , \quad j = 1, 2, \ldots, N \qquad (3)$$

where $\forall_j$, $S_j = \{0, 1, 2, \ldots, K_j\}$ and $r_j : S_j \rightarrow R_+$ is nondecreasing with $r_j(0) = 0$, $\forall_{i,j}$; $g_{ij} : S_j \rightarrow R_+$ with $g_{ij}(0) = 0$ and $b = \{b_1, b_2, \ldots, b_m\}$ 7/, 0.

Let $f(n, \beta)$ be the maximum objective function value of an undominated feasible solution to (1), (2), and (3) in which only the first n variables $(x_1, x_2, \ldots, x_n)$ can be positive and whose resource consumption cannot exceed $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$. When $\sum_{j=1}^{N} r_j(x_j) \leq \sum_{j=1}^{N} r_j(\hat{x}_j)$ and $\sum_{j=1}^{N} g_{ij}(x_j) \geq \sum_{j=1}^{N} g_{ij}(\hat{x}_j)$ with strict inequality holding in at least one of the (M+1) inequalities the feasible solution, $x = (x_1, x_2, \ldots, x_n)$ is said to be dominated by the feasible solution, $\hat{x} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n)$.

For $0 \leq n \leq N$, let R_n be the (domain) set of resource consumption vectors $g_n(k) = [g_{1n}(k), g_{2n}(k), \ldots, g_{2M}(k)]$ of all undominated feasible values of $x_n = k$. Also, for $1 \leq n \leq N$, let F_n be the set of resource consumption vectors β of all undominated feasible solutions $(x_1, x_2, \ldots, x_n)$ to the following subproblem:

$$\max \sum_{j=1}^{n} r_j(x_j) \tag{4}$$

$$\text{s.t.} \sum_{j=1}^{n} g_{ij}(x_j) \leq b_i \quad , i = 1, 2, \ldots, M \tag{5}$$

$$x_j \in S_j \quad , j = 1, 2, \ldots, N \tag{6}$$

Then, as demonstrated in [ ]

$$F_n \subset \{R_n \cup F_{n-1} \cup (R_n \oplus F_{n-1})\} \quad , \quad n = 1, 2, \ldots, N \tag{7}$$

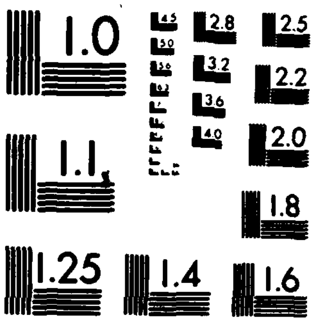
where $(R_n \oplus F_{n-1})^*$ denotes the set obtained by forming all sums of one element of R_n with one element of F_{n-1}. The algorithm proceeds by recursively generating, for all $1 \leq n \leq N$, all feasible candidates for F_n from F_{n-1} and R_n via the following functional equation:

$$f(n,\beta) = \{r_n(k) + f(n-1, \beta-g_n(k) \mid g_n(k) \in R_n, \\ [\beta-g_n(k)] \in F_{n-1}, \beta \leq b\} \tag{8}$$

AD-A130 885 DYNAMIC PROGRAMMING ALGORITHMS AND ANALYSES FOR NONSERIAL NETWORKS PART I..(U) GEORGIA INST OF TECH ATLANTA SCHOOL OF INDUSTRIAL AND SYSTEMS.. A O ESOGBUE 2/2

UNCLASSIFIED JAN 83 ARO-17672.1-MA-H DAAG29-80-G-0010 F/G 12/2 NL

END
DATE
FILMED
8 83
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

and the boundary condition

$$f(0,0) = 0 \tag{9}$$

The implication of the foregoing is that instead of calculating $f(n, \beta)$, $\forall\ \beta \in \Omega$, we only need to calculate it for $\beta \in F_n$ while noting that f_n can be constructed recursively from R_n and F_{n-1}. Furthermore, we can usually eliminate certain elements of either $R_n \cup F_{n-1}$ or $R_n \oplus F_{n-1}$ as being inefficient or infeasible thereby reducing the list length of F_n.

In this way, an M-dimensional search problem is reduced to a one dimensional dynamic programming problem on the sequence of imbedded state spaces $F_0, F_1, \ldots, F_n \subset \Omega$. An algorithm using the above procedure to construct the successive imbedded state spaces and terminate with F_n is illustrated in Table 6.1.

6.3 An Application of ISSA to a Feedforward Loop System

Consider the following nonserial network where each stage has a return expressed as a function of the input variable and each of the input variables has some constraints that need to be met. This is a simple feedforward loop system with constraints.

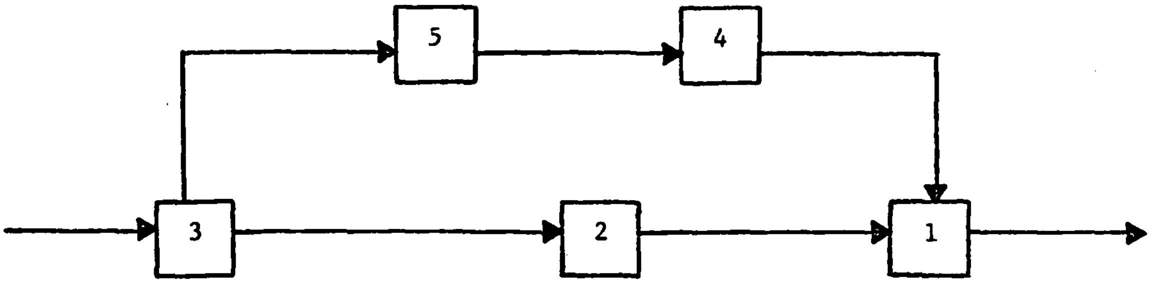


Figure 6.1: An Example of a Nonserial Network (Feedforward Loop)

Suppose we have one constraint for the main serial process and the other for the feedforward branch. That is, the first is for stages 1, 2, and 3 and the second for stages 1, 4, 5, and 3. Then, our problem can be formulated as follows: (10) - (13)

Table 6.1: Steps of the Imbedded State Space Algorithm for the Multidimensional Knapsack Problem

This algorithm may be decomposed into the following steps:

```
Step 1     Set $n=0$, $F_o = \{\beta^o\}$ and $f(o,\beta^o) - 0$, where $\beta^0 = 0$

Step 2     $n \leftarrow n+1$  and $k \leftarrow 0$

Step 3     If $n > N$, stop.

Step 4     $F_{n-1} = \{\beta^o, \beta^1, \ldots, \beta^P\}$ , where $\underline{P} = | F_{n-1} | - 1$

Step 5     $F_n \leftarrow F_{n-1}$

Step 6     $k \leftarrow k+1$   If $k > K$, go to step 2.

Step 7     $g_n(k) = [g_{1n}(k), g_{2n}(k), \ldots, g_{Mn}(k)]$

Step 8     $p \leftarrow 0$

Step 9     If $g_n(k) + \beta^P$ is infeasible go to step 13.

Step 10    If $g_n(k) + \beta^P$ is dominated by some point in $F_n$, go to
           Step 13.

Step 11    $F_n \leftarrow F_n \cup \{g_n(k) + \beta^P\}$
           $f(n, (g_n(k) + \beta^P)) \leftarrow r_n(k) + f(n-1, \beta^P)$

Step 12    Eliminate all the dominated points from $F_n$. i.e., $F_n \leftarrow F_n$ minus
           {all points dominated by $g_n(k) + \beta^P$}

Step 13    $p \leftarrow p+1$

Step 14    If $p \leq P$, go to step 9.
           Otherwise, go to step 6.
```

$$\max \quad r_1(x_1) + r_2(x_2) + r_3(x_3) + r_4(x_4) + r_5(x_5) \tag{10}$$

$$\text{s.t.} \quad g_{11}(x_1) + g_{12}(x_2) + g_{13}(x_3) \leq b_1 \tag{11}$$

$$g_{21}(x_1) + g_{23}(x_3) + g_{24}(x_4) + g_{25}(x_5) \leq b_2 \tag{12}$$

$$x_j \in S_j \tag{13}$$

where $r_j(x_j)$ and $g_{ij}(x_j)$ represent the return and elements of the i-th constraint at stage j respectively.

Let b = (6, 10) and the values of the r_j and g_{ij} functions be tabulated as in Table 6.2.

Table 6.2: Input Data for the Example

x_j	r_1	g_{11}	g_{21}	r_2	g_{12}	g_{22}	r_3	g_{13}	g_{23}	r_4	g_{14}	g_{24}	r_5	g_{15}	g_{25}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	1	2	4	2	0	4	3	4	4	0	3	3	0	8
2	4	2	4	5	3	0	6	6	8	5	0	5	5	0	11
3	5	3	6	9	4	0	7	7	12	8	0	6	6	0	15

Notice the special s:ructure of this problem in which the RHS are constants as opposed to the usual system (see example 4.3) in which they are expressed as functions of the input state and decision variables.

We will now illustrate how the imbedded state space algorithm whose steps were given in Table 6.1 may be used to solve the above nonserial network problem.

Consider the five stage problem as follows:

For Stage 1: $V_1 = \{(0, 0), (1, 2), (2, 4), (3, 6)\}$; $F_0 = \{(0, 0)\}$; $V_1 \circledast F_0 = \{(0, 0), (1, 2), (2, 4), (3, 6)\}$. Since no points are infeasible or dominated we have $F_1 = \{(0, 0), (1, 2), (2, 4), (3, 6)\}$. The following tables are then the result of the computations at the first stage:

FLIST

0	0	0
1	2	2
2	4	4
3	6	5

(M+1)

PERMUTE

2	2	2	1
3	3	3	2
4	4	4	3
0	0	0	4

(M+2)

TRACE

0	0	0
1	1	1
1	2	1
1	3	1

The fourth column appended to the PERMUTE table points to the corresponding entry in the TRACE table

<u>For Stage 2</u>: V_2 = {(0, 0), (2, 0), (3, 0), (4, 0)}; $V_2 \circledast F_1$ = {(0, 0), (1, 2), (2, 4), (3, 6), (2. 0), (3, 2), (4, 4), (5, 6), (3, 0), (4, 2), (5, 4), (6, 6), (4, 0), (5, 2), (6, 4), (7, 6)}. The point (7,6) is infeasible since = (6, 10). The point (2, 0) dominates the point (2, 4). In the same way, (3, 0) dominates (3, 6)and (3, 2), (4, 0) dominates (4, 4) and (4, 2), (5, 2) dominates (5, 6) and (5, 4). Finally, the point (6, 4) dominates (6, 6). So we have F_2 = {(0, 0), (1, 2), (2, 0), (3, 0), (4, 0), (5, 2), (6, 4)} with the following results:

FLIST

0	0	0
1	2	2
2	0	4
3	0	5
4	0	9
5	2	11
6	4	13

PERMUTE

2	3	2	1
3	6	3	2
X	X	X	3
X	X	X	4
4	4	4	5
5	5	5	6
6	2	6	7
7	7	7	8
0	0	0	9

TRACE

0	0	0
1	1	1
1	2	1
1	3	1
2	1	0
2	2	0
2	3	1
2	3	2
2	3	3

For Stage 3: $V_3 = \{(0, 0), (3, 4), (6, 8), (9, 12)\}$, $V_3 \circledast F_2$ contains 28 points, of which 16 are infeasible and 5 are dominated. This leaves $F_3 = F_2 = \{(0, 0), (1, 2), (2, 0), (3, 0), (4, 0), (5, 2), (6, 4)\}$ with the same tables given in Stage 2.

For Stage 4: $V_4 = \{(0, 0), (0, 3), (0, 5), (0, 6)\}$, $V_4 \circledast F_3$ contains 28 points all of which are feasible. Two points are dominated. Finally, we have $F_4 = \{(0, 0), (1, 2), (2, 0), (3, 0), (4, 0), (5, 2), (0, 3), (1, 5), (2, 3), (3, 3), (4, 3), (5, 3), (0, 5), (1, 7), (2, 5), (3, 5), (4, 5), (5, 7), (0, 6), (1, 8), (2, 6), (3, 6), (4, 6), (5, 8), (6, 10)\}$. At the end of this stage, we have the following tables for the FLIST, PERMUTE, and TRACE computations (see next page).

We have used X's in the tables to indicate spaces which, although not currently filled, are useable. Although these appear only in the PERMUTE tables both for stages 2 and 4 in our computation, they should be used in either the FLIST or PERMUTE tables at any stage where their use in dictated.

Notice that the tables begin to look alike as the computational process proceeds to the final stage computation. This is as it should be. Let us now complete the computations for the stage 5, the final stage.

FLIST

1	0	0	0
2	1	2	2
3	2	0	4
4	3	0	5
5	4	0	9
6	5	2	11
7	0	3	4
8	1	5	6
9	2	3	8
10	3	3	9
11	4	3	13
12	5	5	15
13	0	5	5
14	1	7	7
15	2	5	9
16	3	5	10
17	4	5	14
18	5	7	16
19	0	6	8
20	1	8	10
21	2	6	12
22	3	6	13
23	4	6	17
24	5	8	19
25	6	10	21

PERMUTE

7	3	2	1
8	6	3	2
X	X	X	3
X	X	X	4
9	4	7	5
10	5	13	6
11	2	10	7
12	7	21	8
X	X	X	9
13	9	4	10
14	12	14	11
15	10	19	12
16	11	15	13
17	8	22	14
18	13	18	15
19	15	8	16
20	18	9	17
21	16	16	18
22	17	20	19
23	19	12	20
24	20	23	21
2	21	5	22
3	24	6	23
4	22	11	24
5	23	17	25
6	14	24	26
25	25	25	27
0	0	0	28

TRACE

0	0	0
1	1	1
1	2	1
1	3	1
2	1	0
2	2	0
2	3	1
2	3	2
2	3	3
4	1	1
4	1	2
4	1	5
4	1	6
4	1	7
4	1	8
4	2	1
4	2	2
4	2	5
4	2	6
4	2	7
4	2	8
4	3	1
4	3	2
4	3	5
4	3	6
4	3	7
4	3	8
4	3	9

At Stage 5: $V_5 = \{(0, 0), (0, 8), (0, 11), (0, 14)\}$. $V_5 \circledast F_4$ contains 100 points of which 69 are infeasible and 6 are dominated. As a result, we have $F_5 = F_4$ and the resulting tables are the same as in stage 4.

Now we need to find the optimal solution at each stage, The zero in the third column of the PERMUTE table shows that the maximum return is achieved at $b = (6, 10)$ with objective function value 21. To reconstruct the optimal x^*, we go to the TRACE entry in row 28. Here, we find that $x_4^* = 3$. Proceeding to row 9 of the same table, we find $x_2^* = 3$. Finally, in row 3 we have $x_1^* = 2$. All other variables are zero and we have $x^* = (2, 3, 0, 2, 0)$ as the optimal solution and optimal value of 21.

6.4 Analysis of the Imbedded State Space Technique

We will now analyze the storage requirement of the successive imbedded state space technique in solving the following nonlinear knapsack problems.

$$\max \sum_{j=1}^{N} r_j(x_j)$$

$$\text{s.t.} \; \Sigma \, g_{ij}(x_j) \le b_i \quad , \; i = 1, \ldots, M$$

$$x_j \in S_j \quad , \; j = 1, \ldots, N$$

where $S_j = \{0, 1, 2, \ldots, k_j\}$ for all j.

To simplify our analysis we will assume that the maximum permissible value for each variable is taken to be the same, i.e. $k_j = K$ for all j. Then the storage requirement of the technique can be analyzed by considering the following three aspects:

6.4.1) Storage Requirements for the Input Data.

The algorithm needs $r_j(x_j)$, $g_{ij}(x_j)$ and b_i as the input data. Since we have M constraints, N variables and K discretization levels of each variable, the storage requirement for $r_j(x_j)$ and $g_{ij}(x_j)$ becomes $(M + 1)$ KN. To store the available resources the algorithm requires M elements in an array. Hence the storage requirement for the input data becomes $(M + 1) KN + M$.

6.4.2) Storage Requirement for the Three Main Tables.

The three main tables in the imbedded state space algorithm are the FLIST, PERMUTE, and TRACE tables. FLIST table shows the resource used and return for each undominated feasible solution. PERMUTE table gives the best, second best, ... solutions at each stage in conjunction with the FLIST table. Finally, the TRACE table is used in generating optimal solution at the final stage N. It traces from the value of x_N to x_i recursively.

Since the above three tables store only the feasible and undominated (for FLIST and PERMUTE tables) solutions at each stage, the storage requirement actually used is quite different for different problems. Hence, we will analyze the worst case performance of the algorithm. That is, at each stage every point generated is feasible and undominated by the points at the previous stage.

Note that we have K + 1 discretization levels for each variable x_j, j = 1, 2, ..., N and M constraints and one return. At each stage, we have K + 1 times the number of points in the previous stage. By the above assumption, no points are infeasible and dominated. Hence, at the final stage N, each table has $(K + 1)^N$ elements and the storage requirements for the main three tables are $2(K+1)^N (M+1) + (K+1)^N (M+2)$.

6.4.3) Storage Requirement for the Feasibility and Dominance Test.

As shown in the feedforward loop example, the algorithm discards the infeasible or dominated points at each stage. Also, the remaining feasible and undominated solutions need to be sorted for the PERMUTE tables. The itemized storage requirements can be obtained from the computer algorithm. Here, the total storage requirement for the feasibility and dominance tests is given by

$$6N + 4 (M + 1) + (K + 1)^N.$$

Now, the above three storage requirmennts in the imbedded state space technique can be summarized as follows:

a) Storage requirement for the input data = $(M+1)KN+M$

b) Storage requirement for the three main tables $=-2(K+1)^N (M+1) + (K+1)^N (M+2)$

c) Storage requirement for the feasibility and

dominance test = $6N + 4(M+1) + (K+1)^N$

The total is a) + b) + c) or $(K+1)^N (3M+5) + KN(M+1) + 6N + 5M + 4$

6.5 Discussion on the General Nonserial D.P. Network Algorithm and the Imbedded State Space Technique

Let us compare the two approaches using the following feedforward loop system with N stages as our leitmotif.

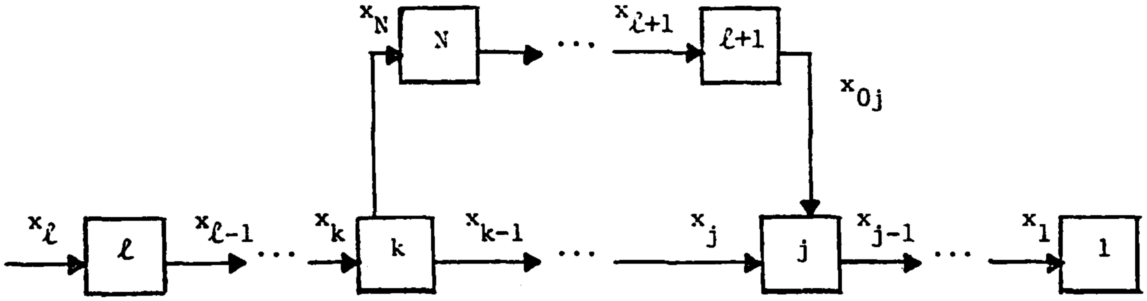


Figure 6.2: A General Nonserial Feedforward Loop Network

The general nonserial D.P. network algorithm solves the following problem:

(P1)
$$\max \sum_{n=1}^{N} r_n(x_n, d_n)$$

$$\text{s.t.} \quad t_n(x_n, d_n) = x_{n-1} \qquad n \neq j, \ell+1$$

$$t_j(x_j, x_{oj}, d_{\ell+1}) = x_{j-1}$$

$$t_{\ell+1}(x_{\ell+1}, d_{\ell+1}) = x_{oj}$$

$$x_n \in X_n, d_n \in D_n$$

On the other hand, the imbedded state space technique was used to solve the following nonlinear knapsack problems:

(P2) $$\max \sum_{j=1}^{N} r_j(x_j)$$

$$\text{s.t.} \sum_{j=1}^{N} g_{ij}(x_j) \leq b_i \ , \quad i = 1, 2, \ldots, M$$

$$x_j \in X_j$$

By comparing the two problems P1 and P2, it is clear that if we delete the decision variables d_n, n = 1, 2, ..., N, then we have the same objective function. However, notice that the two problems are quite different in the structure of the constraints sets. In P1, the constraints are given as the transition function at each stage in the network, while they are given in inequality forms in P2. Also notice that the RHS of each constraint of P1 is not a constant as in P2, but still a variable. Now suppose that the constraint in P2 is changed as N separate ones at each stage, i.e.

$$g_{ij}(x_j) \leq b_i, \quad \text{for} \quad i=1, \ldots, M, \ j=1, \ldots, N.$$

Then, $g_j = (g_{1j}, g_{2j}, \ldots, g_{mj})$ can be considered as a vector transition function at stage j, j = 1, ..., N. However, at the junction stage j in the Figure 6.2, we have two separate transition functions $g_j(x_j)$ and $g_j(x_{oj})$ as opposed to $t_j(x_j, x_{oj})$ in P1 when the decision variables are deleted. Hence, if the transition function at the junction stage j is separable, i.e.,

$$t_j(x_j, x_{oj}) = g_j(x_j) + g_j(x_{oj})$$

then the use of the imbedded state space technique may give a clue to the solution of general nonserial network problems. However, the inequalities in the constraints and the difference in the RHS in the two problems will seem to obstruct the direct use of the technique for the general nonserial network structures. Nevertheless, the ISSI has considerable potential for use in solving large scale state vector dimensional problems in nonserial networks. Its utility will be enhanced by the use of fathoming approaches such as those suggested in the hybrid algorithms described in [21].

References

1. Beightler, C.S., D.B. Johnson and D.J. Wilde, "Superposition In Branching Allocation Problems", Journal of Mathematical Analysis and Applications, Vol. 12, 1965, pp. 65-70.

2. Beightler, C.S. and William Meier, "Design of Optimum Branched Allocation System," Industrial and Engineering Chemistry, Vol. 60, No. 2, February 1968, pp. 45-49.

3. Beightler, C.S. and William Meier, "Branch Compression and Absorption in Nonserial Multistage Systems," Journal of Mathematical Analysis and Applications, Vol. 21, 1968, pp. 426-430.

4. Bellman, R.E.,A.O. Esogbue, and I. Nabeshima, Mathematical Aspects of Scheduling and Applications, Pergamon Press, 1982

5. Bertele, Umberto and Francesco Brioschi, "A New Algorithm for the Solution of the Secondary Optimization Problem in Nonserial Dynamic Programming," Journal of Mathematical Analysis and Applications, Vol. 27, 1969, pp. 565-574.

6. Bertele, Umberto and Francesco Brioschi, " A Contribution to Nonserial Dynamic Programming, Journal of Mathematical Analysis and Applications, Vol. 28, 1970, pp. 313-325.

7. Bertele, Umberto and Francesco Brioschi, "A Theorem in Nonserial Dynamic Programming," Journal of Mathematical Analysis and Applications, Vol. 29, 1970, pp. 351-353.

8. Bertele, Umberto and Francesco Brioschi, Nonserial Dynamic Programming, Academic Press, New York, 1973.

9. Brown, L.G., "Optimization of Nonserial Stochastic Decision Process by Dynamic Programming," Ph.D. Dissertation, University of Arkansas, 1971.

10. Collins, D.C., "Reduction of Dimensionality in Dynamic Programming via the Methods of Diagonal Decomposition," Journal of Mathematical Analysis and Applications, Vol, 30, 1970, pp. 223-234.

11. Collins, D.C. and Lew, Art, "Dimensional Approximation in Dynamic Programming by Structural Decomposition," Journal of Mathematical Analysis and Applications, Vol. 30, 1970, pp. 375-384.

12. Esogbue, Augustine , "Fundamentals of Modern Dynamic Programming," Mimeographed Notes, Department of Operations Research, Case Western University, Cleveland, Ohio, 1970.

13. Esogbue, Augustine, and Marks, Barry, "The Status of Nonserial Dynamic Programming," Management Science, Theory, November, 1972, pp. 350-352.

14. Esogbue, Augustine, and Marks, Barry, "Nonserial Dynamic Programming - A Survey," Operational Research Quarterly, Vol. 25, No. 2, 1974.

15. Esogbue, Augustine, and Marks, Barry, "Dynamic Programming Models of the Nonserial Critical Path-Cost Problem, Management Science, Vol. 24, No. 2, 1977, pp. 200-209.

16. Esogbue, A.O. and N. Warsi, "An Efficient Algorithm for the Solution of a Class of Nonserial Dynamic Programming Problems," Technical Report No. J-83-2, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia

17. Garey, M.R. and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, San Francisco, 1979.

18. Mesarovic, M.O., et al., The Theory of Hierarchical, Multilevel Systems, Academic Press, New York, 1970.

19. Mitten, L.G. and G.L. Nemhauser, "Multistage Optimization" Chemical Engineering Progress, Vol. 59, No. 1, 1963, pp. 52- 60.

20. Morin, Thomas and Augustine Esogbue, "The Imbedded State Space Approach to Reducing Dimensionality in Dynamic Programming of Higher Dimensions," Journal of Mathematical Analysis and Applications, Vol. 48, No. 3, December 1974.

21. Morin, T.L. and R.E. Marsten, "An Algorithm for Nonlinear Knapsack Problems," Management Science, Vol. 22, 1976, pp. 1147-1158.

22. Nemhauser, G.L., Introduction to Dynamic Programming, New York: John Wiley and Sons, 1967.

23. Nemhauser, G.L. and Z. Ullman, "Discrete Dynamic Programming and Capital Allocation," Management Science, Vol. 15, No. 9, May 1969, pp. 494-505.

24. Parker, M.W., "Nonserial Multistage Systems-Analysis and Applications," Unpublished, Ph.D. Dissertation, University of Arkansas, 1969.

25. Parker, M.W. and R.M. Crisp, "Decomposition of Converging Branch Multistage Systems," AIIE Transactions, Vol 2, 1970, pp. 185-190.

26. Wilde, D.J. and Beightler, C.S., Foundations of Optimization, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1967.

27. Wong, Peter and Robert Larson, "Optimization of Tree-Structured Natural Gas Transmission Networks," Journal of Mathematical Analysis and Applications, Vol. 24, 1968, pp. 613-626.

DATE

ILME